M. Stübs

PROGRAMMIEREN MIT TRS-80

ISBN 3-921682- 45–2

1. Auflage 1979
Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime' en RFA.

PROGRAMMIEREN MIT TRS-80

Dieses Buch ist eine Produktion des Ing. W. Hofacker GmbH Verlages. Die Produktion erfolgte unabhängig von TANDY Radio Shack und ihren weltweiten Niederlassungen.

Die Fa. Radio Shack ist eine Division der Fa. TANDY Corporation mit über 7000 Geschäften in 9 Ländern. Zentralbüro in Deutschland, Düsseldorf, Immermannstr.

Literatur- und Quellenverzeichnis

Library 100, eine Software-Bibliothek auf Cassette mit Beschreibung
Hersteller: The Botton Shelf Inc.
P.O.Box 94104
Atlanta, Georgia 30359
Vertrieb in Deutschland: Hofacker Verlag

Technical Manual Radio Shack

80 US, A Journal for the TRS-80 von 80-NW Publishing, Tacoma USA
Deutscher Exklusivvertrieb: Hofacker Verlag

TRS-80 ist ein Warenzeichen der Fa. TANDY Corporation in Forth Worth, Texas.
Deutsche Zentrale:

TANDY Corporation
Immermannstr. 57
4000 Düsseldorf
Tel.: 0211/ 353617/18

Level II BASIC Reference Manual, Radio Shack



Autorenprofil

Martin Stübs, 1946 als Sohn deutsch/englischer Eltern in London geboren, studierte in Hamburg einige Semester Physik, bis er seine eigentliche Neigung entdeckte, die literarische Beschäftigung mit Technik und Wissenschaft. Seit 1974 gehört Stübs als Technik-Redakteur der Redaktion einer Wirtschaftszeitung mit kommunaler Zielsetzung an. Daneben liefert er technisch orientierte Beiträge für andere Publikationen. Seit kurzem gehört sein Interesse auch dem Heimcomputer, der sich nach dem Verständnis des Autors zu einem alltäglichen, ohne zuviele Kenntnisse vom technischen Innenleben verwendbaren Haushaltsgegenstand entwickeln sollte.



Inhaltsverzeichnis

Vorwort

„Vom Standpunkt der Kybernetik betrachtet, ist die Welt ein Ort des Werdens, in der Wissen seinem Wesen nach der Vorgang des Erkennens ist". Diese Worte stammen sinngemäß von Professor Norbert Wiener, dem 1964 verstorbenen Begründer der Kybernetik als Lehre der Information und Steuerung und damit einer der Urväter der elektronischen Datenverarbeitung.
Der Gedanke steht im Zusammenhang mit einer um die Jahrhundertwende eingeleiteten, auch philosophischen Abkehr von einem starren, festgefügten Weltbild, wie es sich auch in dem streng deterministischen Physikmodell Newtons manifestierte. Für die damals neue Vorstellung von der inhärenten Unbestimmtheit aller Vorgänge in der Welt ist vielleicht die Heisenbergsche Unschärfe-Relation das deutlichste Sinnbild. Wieners Verdienst ist es, die zuvor als lästige Unzulänglichkeiten verstandenen Meßungenauigkeiten in der Praxis der Ingenieure als unlösbar mit dem jeweils betrachteten System verbunden erkannt, und bewußt in den beabsichtigten Steuerungsprozeß einbezogen zu haben.
Norbert Wieners wissenschaftliche Arbeit bildet auch eine der Wurzeln für eine heute allen geläufige Vorstellung — nämlich von der überragenden Bedeutung der Information bei allen irgendwie geordneten Systemen. Für unser tägliches Leben bedeutet dies: Im Mittelpunkt unserer Gesellschaft steht der Informationsaustausch, die Kommunikation. Alles spricht heute von „der Energiekrise", — als ob es in der menschlichen Geschichte noch nie Energiekrisen gegeben hätte! Das Gegenteil ist der Fall. Doch „unsere" Energiekrise unterscheidet sich in einem wesentlichen Aspekt von allen vorherigen. Frühere Verknappungen wurden jeweils durch Entdeckung und Ausbeutung eines neuen Energieträgers überwunden — so zum Beispiel zu Anfang der ersten industriellen Revolution in England die Kohle, die das immer knapper werdende Holz ablöste. Dies will uns, die wir erstmals die Begrenztheit der Weltvorräte erfahren, nicht mehr so einfach gelingen

Einen wesentlichen zukunftsträchtigeren Weg als alle Energiesparpläne und Kernkraftwerk-Bauprogramme könnte uns die Mikroelektronik bereits gezeigt haben, ohne daß wir dies schon erkannt hätten: Die schier unglaubliche, noch keineswegs abgeschlossene Verbilligung der Halbleiterfunktionen läßt sich auch verstehen als Abkehr von Energie und Materie überhaupt. Information läßt sich — mit menschlicher

Mühe – beliebig vermehren und, wenn erst einmal erzeugt, fast unendlich oft kopieren. Auf diesem Gebiet wird man bis ans Ende der Menschheit nicht auf Grenzen stoßen.

Nun ist die Erforschung der Informationslehre beileibe nichts Neues. Vom großen Mathematiker und Philosoph Leibnitz haben wir ein mit dem 15. März 1679 datiertes, also gerade 300 Jahre altes Manuskript, in dem die Theorie der binären Zahlen entwickelt wird. Vielleicht könnte man es aber als Fügung des Schicksals deuten, daß elektronische Informationsverarbeitung gerade in dem Augenblick wirklich verfügbar, das heißt in großer Breite anwendbar wird, in dem die Lösung der Probleme unserer Zivilisation nicht mehr einfach noch größerem Einsatz von Energie und Rohstoffen überlassen werden kann.

In großer Breite anwendbar sein heißt natürlich auch, im täglichen Leben jedes Einzelnen eine Rolle spielen. Und was kann da nützlicher sein, als sich aktiv mit der neuen Technik zu beschäftigen, anstatt zu warten, bis sie von selbst auch in das eigene Leben eindringt? Ganz abgesehen von dem praktischen Nutzen oder auch einfach dem Vergnügen, das sich mit der Arbeit mit einem Microcomputer erzielen läßt – diese Maschinen könnten jedem Privatmann die Möglichkeit geben, eine Vorstellung der Mittel und Wege zu gewinnen, auf denen künftige Probleme der Menschheit gelöst werden müssen.

Was Computerfachleuten bereits geläufig ist, könnten, ja sollten Microcomputer jedem vermitteln: Die Denkweise, die zur maschinellen Informationsverarbeitung gehört. Logisches, zielorientiertes Denken ist beim Menschen eng mit der Sprache verknüpft. Dies ist auch ein Grund dafür, daß die einzelnen Bereiche menschlicher Betätigung mit eigenen Fachsprachen verbunden sind. Sie stärken nicht nur das Zusammengehörigkeitsbewußtsein der mit dem Fachgebiet beschäftigten, sondern bilden vielfach auch erst das „Handwerkszeug" zur Formulierung der Gedankengänge, die für das jeweilige Fachgebiet typisch sind.

Gleiches gilt auch für die Computertechnik: Mit dem Lernen einer Programmiersprache erwirbt man mehr als nur die Fertigkeit zur Bedienung einer Maschine. Man macht sich die Denkweise zu eigen, die es erlaubt, Informationsprobleme von Maschinen lösen zu lassen. Und eines ist sicher: Immer mehr solcher Aufgaben werden künftig Maschinen übernehmen – auch im privaten Bereich jedes Einzelnen.

Einleitung

Immer neue Bereiche unseres täglichen Lebens werden von der Microprozessor-Technik beeinflußt, die durch die immer dichtere Packung elektronischer Schaltkreise zu höchstintegrierten Schaltungen möglich wurde. Eines der faszinierendsten Produkte dieser neuen Technik sind die Microcomputer, die als Heimcomputer der elektronischen Datenverarbeitung Anwendungen im häuslichen Bereich erschlossen und sogar die Beschäftigung mit Computern als Hobby ermöglicht haben.

Gegenstand dieses Buches ist der Microcomputer TRS-80 von RADIO SHACK, der in geradezu idealer Weise kostengünstigen Einstig in die aufregende Welt des Elektronenrechners mit der Ausbaufähigkeit zu einem kommerziell einsetzbaren System in sich vereinigt. Angesprochen werden soll der Leser, den weniger das technische Innenleben der Maschine als seine – nicht geringen – Möglichkeiten im häuslichen Bereich, für Schule, Studium, private Finanzen und zum Zeitvertreib interessieren. Um das Feld des echten Heimcomputers nicht zu verlassen, soll dabei im wesentlichen eine Beschränkung auf die bis zu der Preis-„Schallgrenze" DM 3.000,-- erhältliche Geräte-Konfiguration gelten: Die Computereinheit, ausgerüstet mit dem bereits recht kommerziellen BASIC Level II, 4 oder 16K freie Speicherkapazität sowie Sichtgerät, Cassettenrecorder als Massenspeicher und Stromversorgung. Daneben liegt bei der Beschreibung der anschließbaren Peripheriegeräte das Gewicht auf den für private Anwender interessanten Möglichkeiten. Beim Leser vorausgesetzt werden Grundkenntnisse in der Programmiersprache BASIC. Im übrigen bemüht sich das Buch um möglichst einfache, verständliche Darstellungsweise. Das „Computer-Fachchinesisch" soll nur dort verwendet werden, wo es auch beim besten Willen nicht zu vermeiden ist.

Anders als manch anderes Microcomputer-System kann sich der TRS-80 einer ungewöhnlich ausführlichen und gut geschriebenen Betriebsanleitung rühmen. Schon aus Platzgründen wird es sich dieses Buch ersparen, die dort zu findenden Informationen zu wiederholen, wo sich das vermeiden läßt. Allerdings soll auf einige Fehler aufmerksam gemacht werden, die sich in das Werk eingeschlichen haben und dem Anwender im einen oder anderen Fall Kummer bereiten könnten.

„Der äußere Anschein"

Gleich zu Anfang, wenn man stolzer Besitzer eines TRS-80 geworden ist, wird man mit den wichtigsten Eigenheiten des Microcomputer-Systems konfrontiert: Erwartungsvoll öffnet man die Wellpappe-Kiste und findet – für die lange Reise aus den USA nach Deutschland sorgfältig verpackt – nicht ein Gerät wie bei vergleichbaren anderen Fabrikaten, sondern ein wahres Sammelsurium von verschiedenen Dingen. Da ist zunächst einmal der Computer selbst: Ein flaches, 42 cm langes und 21 cm breites Gehäuse, in der eine einzelne Platine untergebracht ist. Auf der Oberseite befindet sich die Tastatur und zwar eine echte Schreibmaschinentastatur mit griffigen Tasten, die einen guten Druckpunkt haben. Wenn man mit deutschen Schreibmaschinen vertraut ist, wird hier allerdings etwas Umgewöhnung fällig . Wie bei großen und kleinen Computern üblich, hat der TRS-80 eine Tastatur nach englisch-amerikanischer Norm. Das heiß, Z und Y sind vertauscht und die Umlaute Ä, Ö und Ü fehlen. Dafür gibt es neben vier nach oben, unten rechts und links weisenden Pfeilen auch noch die üblichen Sonderzeichen #, * und @, die im BASIC (und in einem Fall gerade im RADIO SHACK-BASIC) besondere Funktionen haben. Doch davon später mehr. Eine Anmerkung: # ist im Amerikanischen die Abkürzung für „Nummer", entspricht also unserem „Nr.". In Programmbeschreibungen, Texten u. s. w. taucht dies nicht selten auf.

Neben der normalen Tastatur ist bei neueren TRS-80 Computern ein zusätzliches Zahlen-Tastenfeld angeordnet. Besitzer eines älteren Modells, das darüber noch nicht verfügt, können ihr Gerät zu einem einigermaßen günstigen Preis nachrüsten.

Die einzelnen Tasten arbeiten mit mechanischen Kontakten – leider eine deutliche Schwachstelle des Systems, denn sie neigen sehr stark zum Prellen: Statt des gewünschten Buchstabens bzw. der Zahl stehen plötzlich zwei davon auf dem Bildschirm oder ganz andere. Am schlimmsten ist es, wenn, was nicht selten passiert, irgendwelche über die Tasten ausführbare Funktionen ungewollt eingegeben werden. Gerade beim Programmieren kann sich dies als äußerst lästig erweisen. Als Abhilfe bietet TANDY ein kurzes Maschinenprogramm an, das Prellen unterdrücken soll. (Man hat also zumindest das Problem erkannt). Doch müßte es jedesmal vor Beginn der eigentlichen Tätigkeit eingelesen werden, was ja ebenfalls einen Zeitverlust bedeutet.

Die gleiche Wirkung dürfte zu erzielen sein, wenn man die Kontakte mit Kontaktöl einsprüht. Allerdings muß das Gehäuse dazu geöffnet werden, was sich während der Garantiezeit nicht unbedingt empfiehlt.

An der Rückseite des Computergehäuses findet man (von hinten gesehen) links drei Diodenbuchsen (fünfpolig, nach DIN) und den Schalter für die Stromzufuhr. Rechts ist unter einer abnehmbaren Klappe der 40-polige Bus-Anschluß zur Verbindung mit dem Zusatzteil (expansion interface) sowie der RESET-Knopf angeordnet, den man betätigen kann, wenn ein Programm oder eine andere Tätigkeit des Computers aus irgendeinem Grund steckengeblieben ist. (Häufigster Fall ist hier das mißglückte Laden eines Programms von dem Cassettengerät). Mit den drei DIN-Buchsen wird die Verbindung zur Stromversorgung, zum Bildschirmmonitor und zum Cassettengerät hergestellt, und man kann dem Hersteller nur danken, daß er diese Anschlußssysteme gewählt hat. So werden gewisse einfache Anwendungen bzw. Erweiterungen des Computers für die Anwender zu einer außerordentlich simplen, preisgünstigen Angelegenheit. Davon soll später noch die Rede sein.



TRS-80 mit Drucker

Obwohl der Hersteller sich das wohl nicht so gedacht hat, lassen sich Zusatzgeräte mit Hilfe eines 40-poligen Platinensteckers auch direkt an den Computer anschließen, also unter Umgehung des Zusatzteils, der viele, über den Bedarf des privaten Computerbesitzers meist hinausgehende Funktionen in sich vereinigt und daher auch nicht gerade billig ist. Beispiele dafür sind auch in diesem Buch zu finden. Etwas unglücklich ist die Anordnung der beiden Schalter für Stromversorgung und RESET. Ihr versteckter Platz erklärt sich daher, daß sie ebenfalls direkt auf der Platine montiert sind. Dies entsprang sicherlich dem Wunsch nach möglichst kostensparender Herstellung des Computers. Dies ist zwar auch ganz im Sinne des Käufers, doch hätte man sich doch lieber zwei günstiger plazierte Schalter gewünscht.

Ähnliche Gedankengänge drängen sich auf, wenn man den Video-Monitor (29 cm Diagonale) betrachtet, der als zweiter größerer Bestandteil des Computersystems geliefert wird. Zwar entspricht das Gehäuse-Design ganz der TRS-80-Linie, doch ist die Verwandschaft mit einem tragbaren Fernseher deutlich zu sehen. Sogar der Platz für die Teleskopantenne an der Gehäuserückwand ist noch vorhanden. Und bei der Anpassung an die neue Aufgabe scheint man ebenfalls recht sparsam vorgegangen zu sein:
Bei ungünstiger Aufstellung neigt die Bildschirmeinheit nach einiger Betriebszeit zu recht störendem Brummen.

Die Verbindung zwischen Computereinheit und Bildschirm einerseits und Cassettenrecorder andererseits stellen wie schon erwähnt verschiedene Kabel her. Dazu kommt noch der Anschluß des Computers an sein — getrenntes — Netzgerät. Bildschirm und Rekorder haben außerdem je einen eigenen Netzanschluß — alles in allem ein ziemlicher Kabelsalat, dessen Bändigung anfangs einige Mühe macht. Gerade die drei (!) Netzstecker bereiten einige Probleme: Zum Schutz vor ungewollten Stromunterbrechungen sollten sie so sicher wie nur möglich angeordnet werden, doch ging die Sparsamkeit des Herstellers so weit, daß für die Computereinheit kein Netzschalter vorhanden ist. Nach Gebrauch müssen also vorsichtshalber alle Stecker gezogen werden, und das geht natürlich nicht, wenn sie allzu weit aus dem Wege sind. Gute Dienste leistet hier sicher eine Mehrfachsteckdose mit eingebautem Schalter, die man aus Platzgründen wohl am besten an der Seite des Tisches anbringt, auf dem der Computer seinen Platz hat.

Einen großen Vorteil hat die Bauart des TRS-80 mit getrennten Einheiten übrigens: Sie ermöglicht die freie Anordnung von Tastatur und Bildschirm zueinander – ganz nach den jeweiligen Platzverhältnissen und dem Geschmack des Benutzers. Grenzen setzen hier allerdings die geringe Länge der Verbindungskabel, doch läßt sich dieses Hindernis sehr leicht überwinden: Dazu sind nichts weiter nötig als Verlängerungskabel mit fünfpoligen Steckern und Buchsen nach DIN. Die Wahl dieser in Deutschland so gebräuchlichen Verbindungsart durch TANDY zahlt sich also schon hier für den TRS-80 Besitzer aus.

In gleicher Weise kann man den Cassettenrecorder und die Stromversorgung für den Computer zum Beispiel bequem in einer Schreibtischschublade verstauen. Die Kabel werden dabei am besten unauffällig durch entsprechende Bohrungen in der Rückwand des Schreibtisches geführt. Doch sollte man dies mit dem Cassettengerät erst dann tun, wenn man mit dem Datenaufzeichnen und -überspielen einige Übung gewonnen hat. In einem Schubfach ist das Gerät eben nicht ganz so leicht zu bedienen. Vermeiden sollte es der TRS-80-Freund, Rekorder und Computer-Stromversorgung in unmittelbarer Nähe zueinander anzuordnen: Es ist nicht auszuschließen, daß magnetische Streufelder vom Transformator der Stromversorgung den Recorder stören.

In das Innenleben des Microcomputers scheinen sich die eben beschriebenen, eher äußerlichen Unzulänglichkeiten nicht forzusetzen, und das ist für den ernsthaften Benutzer wohl am wichtigsten. Man kann ziemlich fest auf die einwandfreie Funktion des Gerätes vertrauen. Eine Ausnahme bildet vielleicht die Benutzung des Cassettenrecorders. Davon soll noch die Rede sein - außerdem ist das Cassettengerät streng genommen auch kein Bestandteil des Computers. Auch hinter scheinbar unerklärlichen Fehlern steckt jedenfalls erfahrungsgemäß kein Schaden am Gerät, sondern ein Programmier- oder Bedienungsfehler. Wie noch zu erläutern sein wird, finden sich in der Programmiersprache Radio Shack Level II BASIC, die hier verwendet wird, einige recht versteckte Fußangeln. Ist man allerdings erst einmal mit den Einzelheiten des TRS-80 vertraut, wird man bald seine Qualitäten schätzen lernen, die ihn in manchen Feldern der Anwendung deutlich über andere vergleichbare Systeme hinausragen lassen.

Die Benutzung des Cassettenrecorders
Wie man weiß, ist der Einsatz eines handelsüblichen Cassettenrecorders zur Speicherung größerer Datenmengen der schwächste Punkt aller für den privaten Gebrauch ausgelegten Microcomputersysteme. Erst diese Technik läßt ja die erschwinglichen Preise zu, mit deren Hilfe dieser Anwenderkreis erreichbar wird. Leider macht der TRS-80 dabei keine Ausnahme; im Gegenteil, möchte man fast sagen. Deshalb sollen an dieser Stelle einige Tips folgen, die dem Leser (hoffentlich) einen Teil der zermürbenden, stundenlangen Probiererei ersparen, über die nicht wenige TRS-80-Anwender klagen, wenn es darum geht, mit der Übertragung von Programmen und Daten zwischen Computer und Bandgerät zurande zu kommen.

Zunächst eine Anmerkung: Als dieses Buch entstand, wurde der TRS-80 mit dem Recorder „REALISTIC CTR-80" ausgeliefert. Deshalb beziehen sich die Angaben in diesem Kapitel auf dieses Gerät. Sollte der Leser einen anderen Recordertyp bekommen haben oder aus einem anderen Grund einen anderen verwenden, gelten die genannten Werte natürlich nur sinngemäß.

Am problemlosesten ist naturgemäß das Überspielen von Daten oder Programmen vom Computer auf das Band. Man braucht weiter nichts zu tun, als nach der angegebenen Vorgehensweise eine leere Cassette einzulegen, auf „Aufnahme" zu schalten und den Befehl CSAVE "(Programmname)" einzugeben. Der Programmname (den man jeweils frei wählen kann) muß immer dabei stehen, braucht aber nicht länger zu sein als ein Zeichen (Buchstabe oder Zahl), da der Computer nur das erste Zeichen zur Unterscheidung verschiedener Programme auf einem Band heranzieht. Wie die Anführungszeichen (z. B. bei CSAVE"A") schon vermuten lassen, handelt es sich bei dem Programmnamen um eine Zeichenkettenvariable ("string"-Variable).
Man kann den Befehl also auch so formulieren: A$=X: CSAVE A$. Das hat hier kaum eine Bedeutung, eher schon beim Laden von der Cassette in den Computer, wie im Programm-Anhang des Buches erläutert wird. Achten muß man beim Überspielen auf Band eigentlich nur darauf, daß die Cassetten normalerweise ein Vorlaufband ohne magnetisierbare Schicht haben, so daß man nicht unmittelbar am Bandanfang beginnen kann. Bei der Speicherung von Daten mit dem Befehl PRINT #-1, ... innerhalb eines Programms kann man den nötigen

Vorlauf bis zum Beginn des eigentlichen Tonbandes gleich mit berücksichtigen. Wie in dem entsprechenden Kapitel des Buches erläutert wird, dient dazu der Befehl OUT x,y.

Dringend zu empfehlen ist die Prüfung jedes abgespeicherten Programms mit der Anweisung „CLOAD?", bei der der Computer das gespeicherte Programm mit seinem eigenen Speicherinhalt vergleicht. Allzu leicht kann ein mit viel Mühe geschriebenes Programm durch das unzulängliche Speicherverfahren verlorengehen. Die mit dem Cassettenrecorder arbeitende Methode erreicht als Zugeständnis an die Erschwinglichkeit des Computersystems eben nun mal nicht die Sicherheit üblicher Techniken elektronischer Datenverarbeitung. Man sollte deshalb auch die Empfehlung des Herstellers beherzigen und insbesondere wertvolle Programme mehrmals – vorzugsweise auf verschiedene Bänder – überspielen. Nicht nur kann ein Band verlorengehen oder anderweitig Schaden nehmen und damit auch sein Inhalt. Es geschieht zuweilen auch, daß sich ein ordentlich mit anschließender Prüfung gespeichertes Programm aus unerklärlichen Gründen später nicht mehr laden läßt.

Und damit kommen wir zum eigentlichen Problem der Speicherung von Informationen auf Cassettentonbändern, dem Laden in den Computer. Eine entscheidende Schwierigkeit dabei ist zunächst die richtige Wahl der Wiedergabe-Lautstärke am Cassettengerät. Bei dem Recorder vom Typ CRT-80 klappt das Laden zumeist mit einer Einstellung des Lautstärkereglers (dessen Skala beim Ablesen übrigens einige Mühe macht) in der Nähe der Zahl 4 – vorzugsweise etwas oberhalb davon. So lassen sich auf jeden Fall die Bänder laden, die der Benutzer selbst bespielt hat. Aber auch bei fremden Bändern ist die Einstellung in der Regel richtig – die Empfehlung von RADIO SHACK, die Lautstärke bei fremden Cassetten etwas hochzustellen, kann man außer Acht lassen. Allerdings wird die richtige Lautstärkeeinstellung kritischer ; man darf nicht mehr von der Stellung etwas oberhalb 4 abweichen. Was kann man nun tun, wenn das Laden eines Programms mißlingt? Wenn es sich um ein vom TRS-80-Besitzer selbst gespeichertes Programm handelt, sollte die Lautstärke überprüft und gegebenenfalls richtig eingestellt werden (s. oben). Mißlingt dann ein weiterer Ladeversuch, ist im allgemeinen nichts mehr zu machen - man kann nur noch auf die zweite Aufnahme ausweichen, die man (hoffentlich!) gemacht hat, um solchen Fällen vorzubeugen.

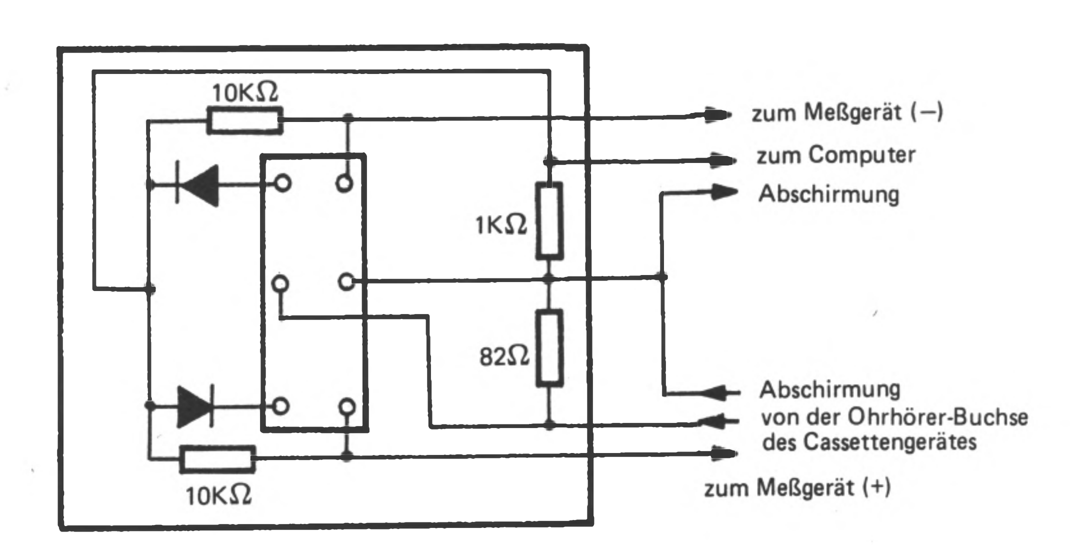
Bei fremden Programmen hilft es oft, die Lautstärke um einen ganz kleinen Betrag zu verstellen. Manchmal sind derartige Bänder allerdings auch mit ganz anderer Lautstärke aufgenommen, und man muß die richtige Einstellung in eventuell wiederholten Versuchen finden. Im Handel bezogene Programmcassetten sind in der Regel elektroakustisch vervielfältigt und zwar manchmal ohne besondere Sorgfalt. Daher dann die Schwierigkeiten beim Laden, die von Störspannungen verursacht werden. (Der Hofacker Verlag liefert ab Herbst 1979 Cassetten mit Ladegarantie) Es empfiehlt sich übrigens aus dem gleichen Grund auch für den Anwender nicht, seine Programme auf diese Weise zu kopieren, zumal der Zeitgewinn gering ist. Für die unverzichtbare Kontrolle der Kopie müßte sowohl das Original als auch die Zweitanfertigung in den Computer gelesen werden (mit CLOAD bzw. CLOAD?).

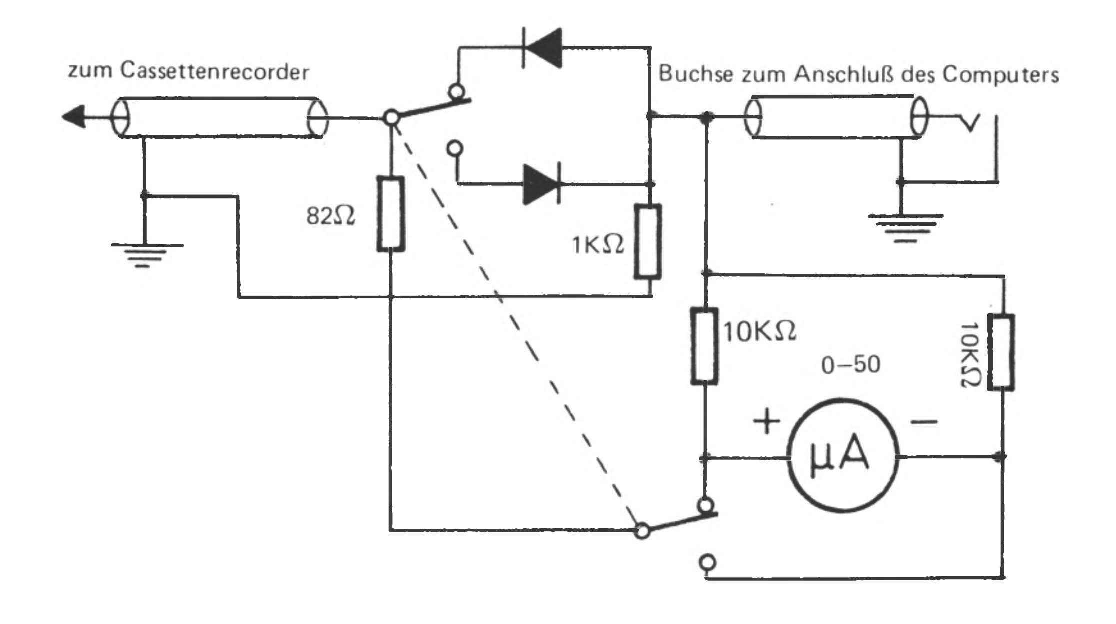
Ein ganz düsteres Kapitel waren bisher leider die für den TRS-80 auf Bändern erhältlichen Maschinenprogramme. Abgesehen von den mit dem Computer in der BASIC-Level II-Version mitgelieferten beiden Bändern für die Übertragung von Daten und Programmen von Level I auf Level II ließen sich in Maschinensprache geschriebene Programme in allzuvielen Fällen absolut nicht laden, und zwar aus demselben Grund: Störungen durch mangelhafte Vervielfältigung. Wie die (leidvolle) Erfahrung zeigt, helfen auch keine Überspielversuche mit verschiedenen Lautstärken. Wenn es bei der normalen Einstellung nicht geht, sollte man sich weitere fruchtlose Bemühungen sparen.

Gibt es nun gar kein Mittel, diese Schwierigkeiten zu überwinden, die den Austausch von Programmen zwischen TRS-80-Besitzern erschweren und im Handel bezogene Cassetten wertlos machen? Doch, das gibt es — in Form eines einfachen, kleinen Gerätes, das man sich notfalls leicht selbst bauen kann und das es erlaubt, den Ladevorgang kontrolliert ablaufen zu lassen und die vom Band abgespielten Signale auf einfache, aber wirksame Weise vor der Einspeisung in den Computer zu filtern.

Signalfilter
Mit diesem kleinen Filter sollte es gelingen, auch Programme zu laden, die sonst als hoffnungslose Fälle gelten müssen:

Teileliste
2 Germanium-Signaldioden
2 Widerstände 10kΩ 5 %
1 Widerstand 1kΩ
1 Widerstand 82 Ω
1 Miniaturschalter 2Xum
1 Einbaumeßgerät, Meßbereich ca. 50 μA
je 1 Stecker und Buchse, passend zum Ohrhörer-Anschluß des Rekorders
ca. 50 cm einadriges abgeschirmtes Kabel





Die Schaltung läßt sich gut in einem kleinen Gehäuse unterbringen, wobei die Anschlüsse des Schalters auch gleich als Lötstützpunkte dienen können.

Das fertige Gerät wird zwischen den Ohrhörer-Ausgang des Rekorders und dem entsprechenden Stecker des Verbindungskabels (schwarz) eingeschaltet. Zum Laden eines Programms läßt man das Band zunächst einmal laufen und stellt fest, bei welcher Stellung des Schalters das Amperemeter am weitesten ausschlägt. In dieser Stellung läßt man den Schalter und regelt dann den Pegel der Signale mit dem Lautstärkeregler des Cassettengerätes auf 15 μA ein. Dann sollte es keine Probleme mehr mit dem Laden des betreffenden Programms (in BASIC oder Maschinensprache) geben.

Es gibt allerdings Fälle, in denen sich die 15μA auch bei voll aufgedrehter Lautstärke nicht erreichen lassen. Dies liegt häufig daran, daß die Justierung des Tonkopfes bei dem Tonbandgerät, das für die Aufnahme verwendet wurde, nicht mit der in dem Gerät des Anwenders übereinstimmt.

Auch dieses Problem läßt sich mit Hilfe des kleinen Hilfsgeräts überwinden: Zuerst probiert man wie gewohnt die Schalterstellung mit der größten Signalstärke aus. Dann läßt man jedoch zunächst die Lautstärke in einer mittleren Stellung und nimmt einen schlanken Schraubenzieher zur Hand (normale Form oder Kreuzkopf). Durch eine kleine Bohrung im Gehäuse des Recorders CTR-80 läßt sich nämlich (bei laufendem Gerät) die Justierschraube des Tonkopfes erreichen, und eine kleine Drehung bringt den Pegel des Signals auf eine brauchbare Höhe. Anschließend kann dann die Einregulierung mit dem Lautstärkeregler erfolgen.

Natürlich ist eine Schwierigkeit mit diesem Vorgang verbunden – nach Neujustierung stimmt die Einstellung des Tonkopfes für die eigenen Programme des Anwenders nicht mehr. Dies kann man vermeiden, indem man sich die Richtung und das Ausmaß der Verdrehung der Justierschraube merkt und den ursprünglichen Zustand nach dem Laden des Programms wieder herstellt. Eine zweite Möglichkeit besteht darin, mit einem eigenen Programm eine Neujustierung vorzunehmen. Auf jeden Fall sollte man ein problembehaftetes Programm möglichst bald selbst auf ein anderes Band abspeichern, um Pegel-Regulierungen, Tonkopfjustierungen u. s. w. für die Zukunft überflüssig zu machen.

Noch einige Ratschläge am Ende dieses Kapitels: Der TRS-80-Besitzer wird in der Regel eine größere Anzahl Programme (oder auch größere Datenmengen) auf einer Cassette speichern und dafür handelsübliche Cassetten verwenden wollen, die pro Bandseite bis 60 Minuten Spielzeit zulassen. Dies ist auch ohne weiteres möglich, doch sollte man unbedingt nur hochwertige Cassetten (mindestens Chromdioxid, besser noch Ferro- oder Ferrochrombänder) verwenden.

Das ist nicht unbedingt eine Frage des vom Band verkrafteten Frequenzumfangs. Die Aufzeichnung erfolgt beim TRS-80 Level II bekanntlich mit 500 Baud, und da kommen schon bei bescheidenen 5kHz, die jedes noch so minderwertige Erzeugnis leicht verkraftet, brauchbare Rechteckimpulse zustande.

Doch ist das Bandrauschen bei besseren Cassetten geringer, und vor allem haben sie keine oder zumindest weniger Unterbrechungen in der magnetisierbaren Schicht (sogenannte dropouts), die natürlich jeden Versuch einer Daten- oder Programmaufzeichnung vereiteln.

Und noch etwas: Da anders als bei Musikaufnahmen schon ein einziger fehlender Impuls die gespeicherte Information wertlos machen kann, sollte man sich bei der Handhabung von Recorder und Cassetten um möglichst weitgehende Sauberkeit bemühen. Cassetten gehören in ihre Boxen und sind am besten in einer Schublade oder an einem ähnlichen Ort aufgehoben.
Es kann auch nicht schaden, sich eine Reinigungscassette zuzulegen und den Tonkopf des Recorders von Zeit zu Zeit damit zu säubern.
Zu dem Cassettengerät liefert Radio Shack auch einen Kurzschlußstecker, dessen Aufgabe es sein soll (so wurde jedenfalls bis vor kurzem ausgeführt), das eingebaute Mikrophon des Geräts für Überspielungen dadurch stillzulegen, daß man ihn in den Mikrophoneingang steckte. Wer das mit dem CTR-80 versucht hat, wird sich sicher gewundert haben, daß anschließend auf dem Band überhaupt nichts zu finden war. Bei diesem Rekorder schaltet der im Mikrophoneingang steckende Kurzschlußstecker nämlich sämtliche Eingänge aus, nicht nur das Einbaumikrophon. Daher kann man den Stecker zwar nicht für den angegebenen Zweck verwenden, dafür aber zum Löschen von Bändern, indem man sie mit eingestecktem Kurzschlußstecker in Aufnahme-Stellung des Recorders durchlaufen läßt. Andererseits ist die beabsichtigte Funktion des Steckers zumindest beim CTR-80 auch nicht erforderlich, da schon die Benutzung des Signaleingangs ("AUX") das Einbaumikrophon sperrt.

Das technische Innenleben des TRS-80

Es ist eigentlich nicht die Absicht dieses Buches, den TRS-80 Freund allzu weit in die technischen Geheimnisse seines Computers einzuführen – schließlich ist es ja gerade einer der großen Vorzüge dieses Systems, unmittelbar für den Benutzer bereitzustehen, der sich nicht auch noch mit dem Innenleben der Anlage beschäftigen will. Doch soll an dieser Stelle wenigstens ein grober Überblick über die Arbeitsweise der einzelnen Komponenten folgen. Solches Wissen kann auch beim praktischen Einsatz des Computers – nicht zuletzt, wenn Zusatzgeräte angeschlossen werden sollen – ganz praktisch sein.

Bekanntlich ist der TRS-80 mit dem Microprocessor Z-80 von ZILOG ausgerüstet – eine Zentraleinheit (CPU = Central Processing Unit) der einen oder anderen Art bildet das Herzstück jedes Microcomputers. Die Z-80-CPU ist wohl noch immer die leistungsfähigste ihrer Art. Im Prinzip verarbeitet sie Informationen mit 8 Bit Breite, ist damit also ein sogenannter 8-Bit-Microprocessor. Doch lassen sich ihre Register zum Teil paarweise zusammenschalten, so daß sie in manchen Aspekten bereits die Leistungsfähigkeit der 16-Bit-Microprocessoren erreicht, die seit kurzem auf dem Markt erscheinen (allerdings nicht mit deren Geschwindigkeit). Die Z-80-CPU ist eine Weiterentwicklung des bereits legendären 8080 von INTEL (inzwischen auch ein SIEMENS-Produkt) und, was die Maschinenprogrammierung betrifft, auch fast vollständig mit ihm „aufwärtskompatibel", d. h., für den 8080 geschriebene Maschinenprogramme laufen auch auf Z-80-Geräten. Doch wurde die Maschinensprache für die Z-80-CPU erheblich erweitert – unter anderem um Anweisungen zum Bewegen ganzer Datenblöcke von einem Speicherort zum anderen, was gerade bei einer Anwendung wie für den hier interessierenden TRS-80 Vorteile bringt.

Gewissermaßen das Rückgrad des Systems bilden ein Adressen- und ein Datenbus (mit "Bus" werden in der Computertechnik die Haupt-Leitungsstränge oder Sammelschienen bezeichnet). Der Datenbus ist 8 Bit breit, d. h. er besteht aus 8 Leitungen, die jeweils auf hohem oder niedrigen Spannungspotential sein können (logisch "1" oder "0"). Auf dem Datenbus laufen Informationen zwischen Zentraleinheit, Speicherregionen und gegebenenfalls den angeschlossenen Zusatzgeräten (im einfachsten Fall ist das nur der Cassettenrecorder) hin und her.

Die Informationen auf dem 16 Bit breiten Adressenbus stammen (normalerweise) stets von der CPU und bezeichnen den Ort, an den Daten auf dem Datenbus geschickt werden oder von dem sie geholt werden. Daneben gibt es noch eine Anzahl von Steuerleitungen, mit denen die einzelnen Funktionen innerhalb des internen Computerbetriebs kontrolliert werden bzw. die eine Zusammenarbeit des Computers mit Zusatzgeräten ermöglichen.

Die Organisation des TRS-80 ist speicherplatzbezogen (memory mapped). Daher werden die einzelnen Komponenten nicht über Ein- und Ausgänge erreicht, sondern sie bilden Teile des Computer-Speicherraums und werden einfach durch Angabe geeigneter Adressen angesprochen. So bilden Tastatur und Videomonitor computer-intern nichts weiter als bestimmte Speicherbereiche. Zum Abrufen bzw. Ausgeben von Informationen werden diese Regionen über den Adressenbus aufgerufen.

Prinzipiell gibt es zwei Speichersorten im TRS-80: Festwertspeicher (ROM - Read Only Memory) und Speicher mit wahlfreiem Zugriff (RAM — Random Access Memory). In 12 K ROM ist das Level II BASIC sowie das Betriebssystem des Computers untergebracht. Ca. 4,7 K vom RAM-Bereich braucht der Computer für eigene Zwecke bzw. sind noch ungenutzt. Der Rest steht schließlich für Programme zur Verfügung. Rund 3,3K bei der „4K"-Version, 15,3K beim Speicherausbau auf "16K".

Der allergrößte Teil des RAM-Speichers besteht aus sogenannten dynamischen RAMs. Sie sind besonders einfach im Aufbau, erfordern aber in regelmäßigen Abständen eine „Auffrischung" ihres Inhalts durch eine Adressierungsansprache. Da die Z-80-CPU Adressen selbst erzeugt (was ja zur speicherplatzbezogenen Organisation des Computers führt) geschieht das RAM-Auffrischen somit automatisch mit, und zwar durch das zur Adressierung gehörige Steuersignal RAS'. Es erfolgt etwa alle 2 Microsekunden. Nur der Bildschirmspeicher (1K) ist statisches RAM, benötigt also keine Auffrischung.

Auch das Tastenfeld wird wie ein RAM behandelt. Hier sind aber keine Halbleiter-Speicherelemente im Spiel, sondern die Tastenkontakte selbst dienen dieser Aufgabe. Beim TRS-80 liefert die Tastatur nämlich

keine fertigen, ASCII-codierten Zeichen- bzw. Funktionssymbole, wie sie vom Computer intern gebraucht werden (ASCII = American Standard Code for Information Interchange). Vielmehr werden die Tasten wie bei einem Taschenrechner über den Adressenbus abgefragt. Die Tastenkontakte stellen direkte Verbindungen zwischen Adressen- und Datenleitungen her, und die resultierenden Daten auf dem Datenbus werden durch Software des ROM-Bereiches in ASCII-Zeichen umgewandelt.

Entsprechend den 1024 Schreibpositionen des Bildschirms ist der zugehörige RAM-Bereich 1K groß. Er wird ständig mit Hilfe eines eigenen Video-Adressengenerators abgefragt und liefert Zeile für Zeile ASCII-codierte Zeichen an den Zeichengenerator. Bei den 7 Bildschirmzeilen pro Zeichen (+5 Zeilen Zwischenraum) auf dem Monitor muß jede Zeile des Speichers 7mal abgefragt werden, um auf dem Bildschirm die volle Schriftzeile zu ergeben: Jedesmal wird nur eine Spur des Elektronenstrahls durch den Zeichengenerator in eine Reihe von Punkten und Strichen aufgelöst. In gleicher Weise werden auch die Lichtpunkte der TRS-80-Bildschirmgraphik erzeugt, nur sind hier alle 12 Bildschirmzeilen einer Schriftzeile im Spiel. Ein Prioritätsbit entscheidet dabei zwischen Zeichen und Graphik.

Immer wenn der Computer (bei Ausführung eines PRINT-Befehls etwa) Zeichen für die Ausgabe auf dem Bildschirm in den Bildschirmspeicher schreiben will, wird die Abfrage-Routine kurz unterbrochen. Dies kann man auf einem beschriebenen Bildschirm in Form kurzer schwarzer Striche erkennen. An der Kürze dieser Striche zeigt sich übrigens die Schnelligkeit, in der dieser Einschreibvorgang vor sich geht.

Den Herzschlag des Computers stellt gewissermaßen der Taktgenerator dar. Er ist mit einem Quarz-Schwingkreis ausgerüstet, der mit einer Frequenz von 10,6445 MHz schwingt. Die Z-80-CPU, die eine Taktfrequenz von maximal 2,5 MHz verkraftet (in der Version Z-80 A sogar 4MHz), läuft im TRS-80 mit rd. 1,774 MHz. Dazu wird die Schwingung des Schwingkreises durch 6 geteilt. Für die Bildung der Video-Ausgabe muß der Schwingkreis zusätzlich eine ganze Reihe verschiedener Taktfrequenzen liefern.

Da sind (unter anderem) erst einmal die zwei Takte, von denen einer oder der andere zur Erzeugung der einzelnen Bildpunkte in den Zei-

chengenerator gespeist wird, je nachdem, ob der Computer pro Zeile 64 oder 32 Zeichen schreibt. Die erste Frequenz ist die Oszillatorfrequenz selbst, die zweite die Hälfte davon, 5,32 MHz. Gewonnen wird sie durch Halbierung der Oszillator-Frequenz. In eine Kette von Frequenzteilern zur Gewinnung der Zeilen- und Bildfrequenz wird ein durch 12-Teilung der Oszillatorfrequenz gebildeter Takt von 887,04 KHz eingespeist. Daraus entsteht zunächst die Zeilenfrequenz von 15,84 KHz (entsprechend 264 Zeilen, davon je 36 am oberen und unteren Bildrand und deshalb nicht sichtbar) und schließlich die Bildfrequenz von 60 Hz.

Wie schon erwähnt, arbeitet der TRS-80 speicherbezogen, die einzelnen Funktionen werden also durch Ansprache der betreffenden Speicherstelle über den Adressenbus aktiviert. Doch kann die CPU auch 256 verschiedene Aus- und Eingänge ansprechen. Was dies für den Anschluß von Zusatzgeräten bedeutet, soll an anderer Stelle erläutert werden. Aber auch intern wird ein Ausgang verwendet, nämlich der mit der Nummer 255. Er dient zwei verschiedenen Zwecken: Zum Datenverkehr mit dem Cassettenrecorder und zum Wechsel in der Bildschirmausgabe zwischen schmaler und breiter Schrift. Letzteres ist wohl deshalb auf diese im ersten Augenblick überraschende Weise gelöst, weil die Erzeugung des Monitorbildes wie bereits angedeutet, ganz unabhängig vom sonstigen Computerbetrieb arbeitet und nur über den gemeinsamen Zugriff auf den Videospeicher mit ihm verbunden ist. Jedenfalls ergeben sich über den BASIC-Befehl OUT und die Funktion INP gewisse Einflußmöglichkeiten, die an anderer Stelle näher erläutert werden.

Die Datenspeicherung auf Cassettentonband geschieht binär, mit einem Impuls für 0 und einem Impulspaar für 1. Der Taktabstand beträgt (bei Level II) 2 Microsekunden, so daß sich eine Übertragungsrate von 500 Baud ergibt. Beim Einlesen werden die auf dem Tonband vorhandenen Impulse zunächst gefiltert, jedoch leider nicht sorgfältig genug, wie manch leidvolle Erfahrung mit vergeblichen Ladeversuchen beweist (darüber ebenfalls an anderer Stelle mehr). Anschließend folgt Gleichrichtung, Verstärkung und Umwandlung in Rechteckimpulse.

Zur Energieversorgung benötigt der TRS-80 drei verschiedene Spannungen: +5V, –5 V und +12Volt. Die zugehörigen Stromstärken betragen 1,2 A, 1 mA und 0,35 A. Die –5V- und +12V-Versorgungen

sind dabei speziell für die dynamischen RAMs vorhanden. Die Regelung der einzelnen Spannungen erfolgt auf der Computer-Platine. Bei +5V und +12V ist eine Kurzschluß-Strombegrenzung eingebaut. Der externe Transformator liefert eine Wechselspannung von 14V, die in der Mitte geerdet ist und zur Gewinnung der +5V- und –5V-Versorgungen dient. Für die 12V-Versorgung wird eine Gleichspannung von 19,8V bereitgestellt.

Das Betriebssystem des TRS-80

Wenn man das Betriebssystem in wenigen Worten charakterisieren wollte, das der TRS-80 Microcomputer in seiner Grundausstattung und mit Level II BASIC zur Verfügung hat, müßte man sagen: Knapp bemessen, aber ausreichend. Zwar ist alles vorhanden, was man für einen einigermaßen komfortablen Betrieb benötigt, um absolute Idiotensicherheit hat man sich dagegen nicht bemüht. Daß eben „nur" 12K ROM-Bereich zur Verfügung stehen, mag dabei einiges erklären. Schließlich ist es ja durchaus im Sinne des ernsthaften Computerfreunds, wenn — wie es hier der Fall ist — zugunsten des BASICs auf allzu ausführliche Statusmeldungen, Fehleranzeigen usw. verzichtet wird. Dementsprechend kann man den TRS-80 im Vergleich zu ähnlichen Systemen durchaus als recht „wortkarg" bezeichnen, wenn er in irgendeiner Weise auch alles erforderliche zum Ausdruck bringt.

Die erste Lebensäußerung des Computers nach dem Anschalten besteht aus der lapidaren Frage: MEMORY SIZE? (Speichergröße?) Jetzt kann man bei Bedarf eine Zahl eingeben, die niedriger ist als die Speichergröße des Rechners (20479 byte bei 4K, 32767 bei 14K) und somit den Rest für Maschinenprogramme reservieren. Danach rafft sich der TRS-80 zu seiner längsten Meldung auf: Mit RADIO SHACK LEVEL II BASIC meldet sich das Betriebssystem. Dann folgt ein READY und in der nächsten Zeile ein 〉, Kennzeichen für die direkte Betriebsart.

In dieser Betriebsart, einer von den vier vorhandenen, erfolgt in der Regel die Kontrolle der verschiedenen Computer-Aktivitäten: Laden von BASIC-Programmen vom Cassettengerät und Speichern der Programme auf Band, Starten und unterbrechen von BASIC-Programmen usw. Auch führt der Computer in dieser Betriebsform fast alle BASIC-Befehle direkt aus und erledigt auch Rechenaufgaben. Vor allem aber dient diese Betriebsart auch zum Programmieren. Für Programmzeilen, Kalkulationen, Befehlsfolgen stehen jeweils 241 Stellen (!) zur Verfügung, allerdings nicht 255, wie manchenorts zu lesen ist.

Mit dem Befehl RUN startet man im allgemeinen ein Programm. Soll der Ablauf an einer bestimmten Zeile beginnen, fügt man die Zeilennummer hinzu. Auch mit dem Sprungbefehl GOTO xxx kann man ein

Programm an einer beliebigen Stelle in Gang setzen. Dabei bleiben anders als bei RUN alle zuvor definierten Variablen, Matrizen usw. erhalten, was für manche Zwecke ganz nützlich ist. Drücken der BREAK-Taste unterbricht ein BASIC-Programm. Mit CONT kann man es an derselben Stelle wieder in Gang setzen, sofern nichts daran verändert wurde. (Variablen-Werte lassen sich wohl ändern!)

Die dritte Betriebsart stellt der sogenannte Edit-Mode dar, in dem man Programmtexte mit einem sehr praktischen Satz von Spezialbefehlen ändern kann. Diese Möglichkeit, durch die der TRS-80 unter seinesgleichen ziemlich hervorsticht, erreicht man durch die Anweisung EDIT, gefolgt von einer Zeilennummer (oder einem Punkt stellvertretend für die gerade vorliegende Zeile, wie später erläutert werden soll). Auch bei einem Syntax-Fehler innerhalb eines Programms schaltet der Computer automatisch auf Edit um.

Schließlich existiert noch eine vierte Betriebsart, der „Monitor"-Betrieb. Dies ist entgegen der Bezeichnung kein Monitor, das heißt ein Hilfssystem zum Programmieren von Maschinenprogrammen, sondern erlaubt nur das Laden von Maschinenprogrammen und ihren Start. Durch den Befehl SYSTEM wird diese Betriebsart erreicht; sie zeigt sich durch ein *? am Anfang der Zeile. Radio Shack bietet übrigens sowohl ein eigentliches Monitor- als auch ein Assembler-Programm auf Band an. Letzteres hat auch einen, dem BASIC-Edit-Betrieb ähnliches Edit-Teil, so daß auch Programme in Assemblersprache leicht bearbeitet werden können.

An dieser Stelle soll schon einmal erwähnt werden, daß sich die einzelnen Kontrollanweisungen wie RUN, CLOAD, EDIT usw. auch in BASIC-Programme einbauen lassen. Davon soll später noch die Rede sein. Aufgrund dieser Möglichkeit lassen sich nämlich einige praktische, kleine Hilfsprogramme bilden.

Keine eigene Betriebsart ist der Datenverkehr mit dem Cassettenrecorder, obwohl manches daran erinnert. Hat man die Anweisung CLOAD, CLOAD? oder CSAVE gegeben bzw. wird INPUT #–1 oder PRINT#–1 in einem Programm ausgeführt, schaltet der Computer jede Kontrollmöglichkeit über die Tastatur ab und konzentriert sich

ganz auf die jeweilige Tätigkeit. Dabei wird der Lauf des Bandantriebsmotors mit einem eingebauten Relais gesteuert. Diese Funktion kann man in einem Programm auch getrennt auslösen, was einerseits interessante Gebrauchsmöglichkeiten für den Cassettenrecorder ermöglicht (z. B. Programme mit auf Band gesprochenem Kommentar), andererseits auch eine einfache Steuermöglichkeit für andere Anschlußgeräte bildet. Doch davon später mehr.

Für die Kontrolle des Datenverkehrs mit dem Cassettengerät stehen nur sehr karge Hilfsmittel bereit: Beim Programmladen erscheinen an der rechten oberen Ecke des Bildschirms zwei Sterne, von denen einer blinken muß, wenn alles klappt. Daß etwas nicht in Ordnung ist, kann man daran erkennen, daß keine Sterne erscheinen oder daß sie erscheinen, aber nicht blinken. Aber auch wenn sie blinken, kann etwas mit dem Laden schiefgegangen sein. Mit einiger Übung kann man das erkennen: Das Blinken wird auffallend langsam.
Unterbrochen wird der Ladevorgang deshalb aber leider nicht von selbst.

Demgegenüber erscheint bei einem Ladefehler, der bei einem Maschinenprogramm passiert, ein C an der Stelle des ersten Sterns: D. h. „check sum error" und bedeutet, daß ein Paritätsfehler aufgetreten ist, das Laden also mißlungen ist. Danach bleibt auch der Cassettenrecorder stehen. Bei BASIC-Programmen geschieht etwas ähnliches nur beim Befehl CLOAD?, der zur Überprüfung eines gespeicherten Programmes dient. Die Reaktion auf einen Fehler ist hier fast ebenso knapp: Der Computer zeigt das Wort BAD (schlecht) und stoppt den Recorder.

Ein Fehler bei der Übernahme von Daten (Zahlen oder Zeichenketten) vom Band wird mit der Fehlermeldung FD ERROR (bad file data, schlechte Speicherdaten) beantwortet. Allerdings gilt sie für verschiedene Fehler, die bei der Datenübernahme aus beliebigen externen Quellen auftreten können.
Absichtlich unterbrechen kann man den Cassettengerät-Betrieb nur durch Drücken der RESET-Taste links hinten am Computergehäuse unter der Klappe.

Noch eine nützliche Einzelheit: Sofern der Computer unter der Kontrolle der Tastatur ist (also nicht während eines laufenden Maschinenprogramms, Datenübertragung vom und zum Recorder etc.) kann seine Arbeit durch Drücken von „SHIFT"% unterbrochen werden. Zum Beispiel auch beim Listen von Programmen auf dem Bildschirm, was sehr praktisch ist, wenn man sich den Programmtext Stück für Stück anschauen will. Bei einem Druck auf eine beliebige Taste setzt der Computer seinen Betrieb dann wieder fort.

Ein wesentlicher Vorteil der BASIC-Interpreter, mit deren Hilfe BASIC-Programme intern Zeile für Zeile in Maschinensprache übersetzt und ausgeführt werden, ist die Möglichkeit zur sofortigen Fehlererkennung und -anzeige. Die Anzeigen fallen beim TRS-80 wiederum sehr knapp aus: Je nachdem, ob der Fehler innerhalb eines Programms auftritt oder nicht, werden vor die Meldung ERROR IN (Zeilennummer) bzw. nur ERROR lediglich zwei Buchstaben als Abkürzung gestellt. Was sie bedeuten, muß man eben lernen (oder jeweils im Handbuch nachschauen). Abgesehen davon wird die Fehlererkennung aber sehr weit getrieben: Es gibt 23 verschiedene Meldungen. Und die Fehlerroutinen sind darüber hinaus zu einem wirkungsvollen Instrument bei der BASIC-Programmierung ausgebaut! Ein ganzer Satz von Befehlen und Funktionen —ON ERROR GOTO, ERR, ERL, ERROR, RESUME — steht zur Verfügung, um „Fehler" als Teil des jeweiligen Programms zu verarbeiten, ja, sie absichtlich ins Programm einzubauen.

Das RADIO SHACK Level II BASIC

Ein weiteres wesentliches – wenn nicht das wesentlichste – Merkmal für die Leistungsfähigkeit eines Microcomputer-Systems von der Art des hier behandelten ist die Qualität des in Form von Festwertspeichern (ROMs) eingebauten BASIC. Nachdem diese Sprache ja im Gegensatz zu den übrigen bekannten höheren Programmiersprachen leider nicht genormt ist, hat ziemlich jeder Anbieter auf dem Markt auch sein eigenes BASIC, und so auch Radio Shack. Im wesentlichen unterscheiden sich die verschiedenen Spielarten zwar nur in den Abkürzungen für sonst gleiche Befehle. Doch muß jeder Microcomputer-Hersteller aus der Gesamtheit der zur vollständigen BASIC-Sprache gehörenden Befehle auswählen, um mit dem immer beschränkten Speicherraum auszukommen. Nach Möglichkeit sollten dazu noch einige, auf die speziellen Eigenheiten und Vorzüge des jeweiligen Systems zugeschnittenen Anweisungen kommen.

Wie schon erwähnt, sind im TRS-80 12K für das Level-II-BASIC reserviert. Nachdem das Level I eine einfache, auf den Anfänger zugeschnittene, aber für den Speicherraum von (nur) 4K doch schon recht leistungsfähige BASIC-Version darstellt, ist die höhere Stufe dem dreifachen Speichervolumen entsprechend ungleich leistungsfähiger und komfortabler. Geboten werden ganz ungewöhnlich weitgehende Programmierhilfen. Darüber hinaus ist die Zielrichtung über das Hobby- und Heimgerät hinaus auf den ernsthaften, kommerziellen Einsatz des erweiterten Systems nicht zu verkennen. Neben den Funktionen, die (leider erst) mit dem Geräteanschluß-Zusatzbauteil „Expansion Interface" zugänglich werden, fehlen eigentlich nur noch die Befehle zur Matrix-Behandlung, wie MAT ZER, MAT INV u. s. w. Vom Expansion Interface soll weiter hinten noch die Rede sein.

Programmieren mit dem TRS-80

Wie bereits ausgeführt, muß für Maschinenprogramme, die neben BASIC-Programmen im Speicher untergebracht werden sollen (etwa um sie vom BASIC-Programm aus anzusprechen) entsprechender Speicherplatz reserviert werden. Dazu hat der Programmierer jeweils einmal nach Anschalten des Computers die Möglichkeit.
Auf dem Bildschirm erscheint in diesem Augenblick nämlich die Frage „MEMORY SIZE?".

Will man Speicherraum freihalten, muß man die gewünschte Anzahl Speicherstellen (Bytes) von der höchsten Adresse des jeweiligen Computers abziehen (20479 bei der 4K-Version, 32767 bei der 16K-Version) und diese Zahl eingeben. Ansonsten einfach „Enter" drücken.

Es gibt eine recht einfache Methode, an diesen Punkt zurückzukommen, ohne den Computer aus- und nach entsprechender Wartezeit wieder anschalten zu müssen: Dazu muß man „SYSTEM" eingeben, um in die „Monitor-Betriebsart" zu kommen und danach einfach "/" und "Enter" drücken. Der Computer versucht dann einen Sprung zu einem nicht vorhandenen Maschinenprogramm und landet in der Anschalt-Routine, die mit „MEMORY SIZE?" endet. Im Normalfall wird man jedoch keinen Speicherraum reservieren wollen. Also drückt man nur "Enter", und der Computer meldet sich mit "RADIO SHACK LEVEL II BASIC" sowie "READY" und einem " 〉 " als Bereitschaftszeichen. Nun kann das Programmieren beginnen.

Und hier bietet der TRS-80 bereits die erste praktische Hilfe an: Mit dem Befehl "AUTO" kann man automatische Zeilennumerierung auslösen. Zwei nachgestellte, durch ein Komma getrennte Zahlen geben dabei Anfangs-Zeilennummer und Schrittweite an, z. B. AUTO 10,20 führt zu den Zeilennummern 10, 30, 50,.... Fehlende Zahlenangaben ersetzt der Computer durch die Zahl 10, d. h. er fängt bei 10 an bzw. zählt in Zehnerschritten oder beides, je nachdem, ob die erste Zahl die zweite Zahl oder beide fehlen.

Da wir gerade bei den Programmzeilen sind: Der TRS-80, Level II läßt Programmzeilennummern zwischen 0 und 65629 zu, also insgesamt 65530 Zeilen. Das sind sicher mehr als genug für jedes denkbare Programm, und man hat in jedem Fall die Möglichkeit, über die üblichen Zehner-Schritte hinauszugehen und sein Programm durch geschickte Wahl der Zeilenzahlen zu gliedern. So lassen sich einzelne, zusammenhängende Programmteile mit einer „runden" Anfangszeilen-Nummer kennzeichnen (z. B. 1000, 2000, 3000 u. s. w.). Auch sollte man das Programmieren mit einer nicht zu kleinen Zeilenzahl beginnen und auch größere Abstände zwischen einzelnen Programmteilen lassen. Erfahrungsgemäß gibt es besonders an solchen Stellen oft nachträglich etwas zwischenzuschieben, bzw. voranzustellen, und dabei kann man leicht in Schwierigkeiten kommen.

Die Folge ungenügender Zeilenzahl-Abstände wäre dann umständliches Neuschreiben bereits vorhandener Zeilen oder ein Programm, das durch unnötige Sprungbefehle ("GOTO ...") an Übersichtlichkeit verliert.*

Wie auch in der Programmieranleitung des Computers ausgeführt, kann man Speicherplatz sparen und die Programmausführung durch den Computer beschleunigen, wenn man möglichst viele Anweisungen jeweils zu einer Zeile zusammenfaßt. Mit einer größten Zeilenlänge von 255 Stellen eröffnet der Computer hier auch ungewöhnlich große Möglichkeiten. Allerdings empfiehlt es sich nicht, beim Programmieren gleich von Anfang an in jede Zeile soviel wie nur möglich hineinzupacken. Erst später stellt man dann nämlich fest, daß eine Anweisung, zu der mit einem "GOTO"-Befehl gesprungen werden soll, irgendwo in der Mitte einer Zeile steht und damit mit einem Sprung nicht erreichbar ist. Zudem leidet die Übersichtlichkeit des Programms, was recht lästig werden kann, wenn man später irgendwelche Änderungen vornehmen will. Und bedingte Anweisungen ("IF ... THEN") müssen im allgemeinen sowieso am Ende einer Zeile stehen, wenn man scheinbar unerklärliche Fehler vermeiden will. Am besten ist es, wenn man beim Schreiben des Programms zunächst nur inhaltlich zusammengehörige Befehle in jeweils einer Zeile zusammenfaßt – z. B. zusammenhängende "PRINT"-Anweisungen oder Variablen-Bestimmungen (A=3, B=0 u. s. w.). Wenn man sicher ist, daß das Programm vollständig ist und keine Fehler mehr enthält, kann man damit beginnen, benachbarte Zeilen zusammenzufassen. Dabei muß darauf geachtet werden, daß Anweisungen, die mit einem Sprung erreicht werden sollen, am Anfang stehen und bedingte Anweisungen die Zeile beschließen. Diese beiden Bedingungen beschränken meist die erreichbare Zeilenlänge, doch man sollte die Zusammenfassungen auch sonst nur soweit treiben, wie es sich im Interesse der Übersichtlichkeit noch vertreten läßt.

Weitergehen sollte man nur, wenn bei einem sehr langen oder mit vielen Zahlen bzw. Ziffernreihen (Strings) operierenden Programm der Speicherplatz kanpp wird. Deutliche Einsparungen an Ausführungszeit ergeben sich außerdem nur, wenn Zeilen innerhalb von Programmschleifen zusammengefaßt werden, wo dieselben Anweisungen viele Male hintereinander ausgeführt werden.

* Seit kurzem bietet TANDY dazu auch ein Maschinenprogramm zur Umnumerierung von Zeilen an. Die Verwendung kostet aber wiederum zusätzliche, vermeidbare Mühe.

An weiteren Befehlen, die zu diesem Thema gehören, kennt das Level-II-BASIC außer dem üblichen "LIST" mit seinen Variationen und dem bereits erwähnten "EDIT", das die Bearbeitung einzelner Zeilen mit einem Satz eigener Unteranweisungen erlaubt, noch drei weitere Befehle: "DELETE", "TRON" und "TROFF".

DELETE dient zum Entfernen von Zeilen aus einem Programm und hat ähnlich dem LIST-Befehl drei Versionen: Mit einer nachgestellten Zeilenzahl oder einem PUNKT (.) entfernt es die angegebene bzw. die gerade vorliegende Zeile aus dem Programm, mit einem Zeilenbereich (z. B. 20 – 80 oder auch – 80, d. h. alle Zeilen bis Zeile 80) die darin vorhandenen Zeilen. Allerdings müssen die genannten Zeilen (bei dem Zeilenbereich die letzte, in unserem Beispiel also Zeile 80) auch tatsächlich im Programm enthalten sein, sonst erfolgt eine Fehlermeldung – eine Einschränkung, die für den LIST-Befehl nicht gilt. Echten Nutzen bringt DELETE eigentlich nur, wenn man es mit einem Zeilenbereich anwendet, weil es einem dann das sonst erforderliche Eintippen sämtlicher infrage kommender Zeilennummern erspart. Sonst ist man für das Zeilenlöschen mit dem üblichen Zeilenzahl-Eintippen ebensogut bedient, wie mit DELETE.



TRON und TROFF gehören zusammen und stellen eine recht wirksame Hilfe für die Fehlersuche in Programmen dar. TRON veranlaßt den Computer, bei der Abarbeitung jeder Programmzeile die Zeilenzahl in eckigen Klammern auf dem Bildschirm zu schreiben. TROFF hebt die Anweisung TRON wieder auf. Man kann vor dem Start eines Programms TRON eingeben und damit die Anzeige aller Zeilenzahlen im Ablauf der Programmverarbeitung erreichen. Besonders wenn im Programm Schleifen enthalten sind, hat dies jedoch meist zur Folge, daß der ganze Bildschirm mit Zahlen vollgemalt wird und die im Programm vorgesehenen Anzeigen so durcheinandergebracht werden, daß eine Fehlersuche garnicht mehr möglich ist. Die Anzeige der Zeilenzahl erfolgt nämlich da, wo sich der Läufer (Cursor) gerade befindet, d. h. in der Regel nach der zuletzt angezeigten Zeilenzahl, wenn auf dem Bildschirm gerade nichts neues geschrieben wurde. Als Abhilfe kann man an jeder Programmzeile die Anweisung PRINT CHR$(28) anfügen, was den Läufer immer wieder in die linke obere Bildschirmecke befördert. Dann werden die Zeilennummern nur dort angezeigt, doch ist dies ein recht mühsames Verfahren und kann das eigentliche Programm leicht durcheinanderbringen. Besser ist es, der TRS-80-Anleitung zu folgen und TRON sowie TROFF als Anweisungen im Programm zu verwenden. Vor und nach einer zu überprüfenden Zeile eingesetzt, zeigen die beiden Befehle nur deren Nummer und die Nummer der TROFF-Befehlszeile an, was das übrige Bild auf dem Bildschirm nicht allzusehr stören dürfte.

Unverzichtbarer Bestandteil jeder Programmierarbeit am TRS-80 ist natürlich die "EDIT"-Betriebsart mit ihrem Satz von 14 eigenen Befehlen. Allerdings erfordert die nutzbringende Anwendung der vielfältigen Möglichkeiten zur Veränderung vorhandener Programmzeilen einige Übung, und auch wer BASIC bereits beherrscht, muß hier noch einmal etwas Mühe zum Erlernen der Anweisungen aufbringen. Doch der Aufwand lohnt sich, und mit wachsender Erfahrung wird man die Vor- und Nachteile der einzelnen Befehle für die verschiedenen Anwendungszwecke bald abschätzen lernen und immer kompliziertere Eingriffe vornehmen, ohne die betreffende Programmzeile neu schreiben zu müssen.

Während sich der Computer in der EDIT-Betriebsart befindet, ist allerdings erhöhte Aufmerksamkeit bei der Bedienung dringend zu empfeh-

len. Fehler, die durch Unachtsamkeit oder durch das ärgerliche Tastenprellen entstehen, werden nicht verziehen, und als Folge davon kann die in Bearbeitung befindliche Zeile verstümmelt werden oder manchmal sogar einfach verschwinden. Am schwerwiegendsten ist es, wenn als Folge einer mißglückten Redigier-Arbeit die Zeile zwar äußerlich intakt aussieht, im zugehörigen Befehlscode aber Fehler zurückgeblieben sind. Dann reagiert der Computer mit einer unerklärlichen Fehlermeldung und es hilft nur vollständiges Neuschreiben der Zeile – genau das, was man mit dem Einsatz der EDIT-Möglichkeit verhindern wollte. Deshalb sollte man den EDIT-Vorgang am besten sofort abbrechen, sobald sich ein Fehler zeigt, und von vorne beginnen.

Über die Fehlererkennung und -anzeige durch den TRS-80 ist im Kapitel über das Betriebssystem schon ausführlicher gesprochen worden. An dieser Stelle bleibt vielleicht noch zu sagen, daß für den ungeübten Programmierer die Anzeige von Art und Programmzeile eines Fehlers zunächst das wichtigste Hilfsmittel beim Erstellen von Programmen überhaupt ist. Wenn man die schlichten Tippfehler außer Acht läßt, werden derartige Pannen mit wachsender Erfahrung jedoch seltener, so daß die Anzeigen für die Programmierarbeit an Bedeutung verlieren. Eine Ausnahme machen vielleicht die Meldungen OM (out of memory) und OS (out of string space), die anzeigen, daß die gesamte Speicherkapazität bzw. der für Zeichenreihen reservierte Platz belegt ist. Diese beiden Anzeigen kann man dazu verwenden, ein umfangreiches Programm auf den vorhandenen Speicherraum zuzuschneiden und für die Zeichenreihen den kleinsten, gerade noch ausreichenden Platz zu reservieren.

Als einigermaßen mühsam stellt sich heraus, den Sinn der 23 verschiedenen Fehlerabkürzungen zu erlernen, so daß man nicht ständig in der Anleitung nachschauen muß. Allerdings treten besonders einige wenige Fehler immer wieder auf, deren Abkürzungen man schnell gelernt hat.

Ganz praktisch ist die Tatsache, daß der Computer bei einem „Rechtschreibefehler" (SN = syntax error) gleich von selbst mit der fehlerhaften Zeile in die EDIT-Betriebsart geht. In diesem Fall muß der Fehler ja durch Änderungen an derselben Zeile behoben werden, in der er auftrat. Für andere Fehlertypen gilt dies nicht unbedingt.

Als weitere Programmierhilfe wäre schließlich auch noch die Anweisung "ERROR" zu nennen. Mit einer Zahl zwischen 1 und 23 versehen, simuliert der Befehl einen der 23 erfaßten Fehlerarten, die außer der Abkürzung als Kennzeichen auch eine laufende Nummer haben. Der ganze Vorgang dient in erster Linie zum Testen von ON ERROR GOTO-Routinen, die weiter hinten im Buch behandelt werden. Große Bedeutung hat ERROR nicht, da sich die Fehlerroutinen auch ganz gut ohne besondere Hilfsmittel überprüfen lassen.

Numerische Variable

Im Gegensatz zu den anderen vergleichbaren Microcomputern wartet der mit BASIC Level II ausgerüstete TRS-80 mit vier verschiedenen Variablen-Typen auf, davon drei unterschiedliche numerische Variable. Die für andere Systeme typische neunstellige Genauigkeit der nicht ganzzahligen Variablen ist dabei gewissermaßen aufgespalten in eine sechsstellige und eine 16-stellige (!) Version. Die in den verschiedenen BASIC-Versionen ziemlich einheitlich vorhandenen Funktionen, die es hier auch gibt, lassen sich aber nur mit den sechsstelligen und den ganzzahligen Variablen anwenden. Bei den 16-stelligen Zahlen sind nur die vier Grundrechenarten +, –, x und / möglich.* Von diesen Besonderheiten abgesehen, unterscheidet sich das Radio Shack Level II BASIC nicht von anderen Versionen der Computersprache. Die ganzzahligen Variablen umfassen den Bereich von –32768 bis + 32767, die beiden anderen Typen den Bereich von ca. – 1,7 x 10^{38} bis + 1,7x10^{38}.

Ohne Zweifel steckt hinter dieser Aufteilung der Variablen-Genauigkeit in zwei Stufen auch die Absicht des Herstellers, das Rechnersystem für weitreichendere, kommerzielle Einsatzgebiete geeignet zu machen, wo z. B. für Buchhaltung oder ähnliches 16-stellige Genauigkeit, aber keine höhere Mathematik gebraucht wird. Für die komplizierteren Rechenvorgänge ließ die gegebene Leistungsfähigkeit des BASICs dann wohl nur noch die sechs Stellen der „einfachen" Variablen zu. Der Schnelligkeit des Rechners kommt diese geringere Genauigkeit jedenfalls zugute, und für die meisten Anwendungsfälle dürften die sechs Stellen sicher auch mehr als genügen.

Wie schon erwähnt, verfügt der TRS-80 über die üblichen Funktionen, die aber auf sechs Stellen Genauigkeit beschränkt sind. Als Besonderheit gibt es hier zwei verschiedene Funktionen, die den ganzzahligen Anteil einer nicht ganzzahligen Nummer liefern: INT(x) und FIX(x). INT(x) rundet immer ab, FIX(x) schneidet einfach die Stellen hinter dem Komma ab. Bei negativen Zahlen gibt das einen Unterschied: INT(–3.4)=–4, aber FIX(–3.4)=–3.

* Allerdings sind von Tandy Unterprogramme in Maschinensprache erhältlich, die Wurzelziehen u. s. w. mit 16-stelliger Genauigkeit erlauben.

Obwohl der Rechner intern sieben Stellen verwendet und dann eine auf sechs Stellen gerundete Zahl anzeigt, hapert es ein wenig mit der Genauigkeit der kalkulierten Funktionswerte. Beim Potenzieren ist das noch am leichtesten zu erkennen: 3 ↑ 4 ergibt z. B. 81,0001 statt richtig 81, 3 ↑ 6 729,001 statt 729. Auf die Richtigkeit der letzten Stelle sollte man sich also zumindest bei solchen Werten nicht allzusehr verlassen.

Zeichenreihen (Strings)
Einen weiteren Beweis für seine Leistungsfähigkeit gibt der TRS-80 mit den Möglichkeiten zur Behandlung von Zeichenreihen, „Strings". Die größte Länge eines Strings beträgt stattliche 255 Zeichen. Außer den in anderen Microcomputern auch vorhandenen Funktionen zum Manipulieren von Zeichenreihen gibt es noch die Funktion STRING$ (n, x). Sie bildet eine Reihe gleicher Zeichen, wobei n die Anzahl der Zeichen ist (bis zu 255) und x das Zeichen angibt, entweder das Zeichen selbst in Anführungszeichen, oder die entsprechende Zahl des ASCII-Codes. Dies ist vor allem für graphische Zwecke sehr praktisch.

Die Zahlen 0 bis 31 des Codes sind, wie bereits erwähnt, zum Teil mit Funktionen für die graphische Ausgabe belegt, die Zahlen ab 128 mit den 64 graphischen Zeichen und bis zu 63 Leerstellen. So lassen sich mit Hilfe der beiden Funktionen CHR$ (n), STRING$(n,x) und sämtlichen Buchstaben, Zahlen, Zeichen und Funktionen beliebige Figuren, Schriftbilder u. s. w. als Zeichenreihen zusammenstellen und durch Aufruf eines Zeichens auf dem Bildschirm darstellen. Der Phantasie des Programmierers sind hier kaum noch Grenzen gesetzt.

Da der zum Speichern von Strings reservierte Raum durch den Befehl CLEAR n vor dem Start eines Programms oder im Programm selbst angegeben werden muß (sofern er 50 Byte Speicherstellen übersteigt), gibt es auch die Anweisung FRE(A$), die ähnlich dem MEM für den gesamten Speicherraum den für Zeichenreihen noch zur Verfügung stehenden Platz aufruft.

Das "A$" ist dabei nur eine Pseudovariable ohne praktische Bedeutung. Es muß allerdings eine String-Variable oder ein String sein. Mit einer Zahl oder einer numerischen Variablen liefert FRE wie MEM den gesamten Speicherraum.

Zahlen- und Zeichenfelder (Matrizen)
Mit der Anweisung DIM(....) lassen sich in der BASIC-Sprache bekanntlich Zahlenfelder und Zeichen definieren, die auch mehrdimensional sein können, das heißt, durch mehr als eine Kennzahl bestimmte Elemente enthalten. Beim Radio Shack Level II BASIC ist die Zahl der Dimensionen nicht begrenzt, doch auch mit 16K freier Speicherkapazität ist es kaum sinnvoll, über die Dimension 3 hinauszugehen. Allzu-

schnell verbraucht sich der Speicherraum, und man kann weniger Zahlen (oder Programmlänge) unterbringen, als mit mehreren, kleiner dimensionierten Feldern. Die vier-dimensionale Matrix X(7,7,7,7) aus Zahlen einfacher Genauigkeit (sechs Stellen) läßt sich nicht einmal mehr im sonst leeren 16-K-Speicher unterbringen, wohl aber sieben dreidimensionale Matrizen A(7,7,7) bis G(7,7,7), die zusammen eben soviele Zahlen enthalten, nämlich 2401. Außerdem bürden hochdimensionierte Matrizen dem Computer viel zusätzliche Arbeit auf, und er wird entsprechend langsamer.
Die bei Microcomputern allgemein nicht verfügbaren Matrix-Befehle findet man beim TRS-80 auch nicht — das wäre bei dem Preis der Anlage wohl auch etwas viel verlangt. Immerhin wird man wenigstens die Anweisung MAT X = ZER, die alle Elemente der Matrix X gleich Null setzt, nicht allzusehr vermissen. Dies steht nicht selten am Anfang eines Programms und das Level II BASIC setzt bereits mit dem Startbefehl RUN (oder mit CLEAR) alle Variablen auf Null.

Automatisch werden übringens allen im Programm auftauchenden Feldern je Dimension elf Elemente zugeordnet (Nummern 0 bis 10). Falls man nicht mehr Elemente braucht oder aus Platzgründen die betreffende Matrix kleiner dimensionieren will, kann man sich die DIM-Anweisung ganz sparen.

Variablen-Namen
Bezeichnen kann man die Variablen mit einem oder zwei Buchstaben oder mit einem Buchstaben und einer Zahl. Man kann auch längere Bezeichnungen verwenden, sie müssen jedoch immer mit einem Buchstaben beginnen. Sinn hat das nur als Orientierungshilfe beim Programmieren, denn der Computer hält verschiedene Variable allein an den ersten beiden Stellen auseinander. Ansonsten macht es bloß mehr Mühe und kostet zusätzlichen Speicherraum. Insgesamt ergeben sich um die 900 verschiedene Namen für jeden Variablen-Typ und zusätzlich ebensoviel für jeden Matrix-Typ; sicher mehr, als man selbst im kompliziertesten Programm brauchen wird. Die sich rechnerisch aus den obigen Angaben ergebende Zahl von 962 Namen pro Variablen-Typ verringert sich dadurch etwas, daß Namen nicht verwendet werden können, die BASIC-Wörter enthalten.

Bestimmung der Variablen-Typen
Entsprechend der größeren Zahl verschiedener Arten von Variablen gibt es schon theoretisch auch eine größere Menge von Möglichkeiten, sie festzulegen, zwischen ihnen zu wechseln u. s. w. Und hier hat der Hersteller eine Menge getan, um dem Programmierer die Mittel zum Erstellen leistungsfähiger und dabei platzsparender Programme in die Hand zu geben. Auf vielfältige Art kann man dementsprechend insbesondere mit den Zahlenwerten verschiedener Genauigkeit umgehen. Es gibt aber auch ebensoviele Möglichkeiten, Fehler zu machen, die dann in der Regel unbemerkt zu falschen Ergebnissen einer Computerberechnung führen. Es gehört schon einige Übung dazu, unter den gegebenen Möglichkeiten die für den speziellen Fall günstigste herauszusuchen, wenn man mit Zahlen verschiedener Genauigkeit arbeiten will, und dabei die Fallen umgehen, die sich in den manchmal recht verzwickten Regeln verbergen.

Es gibt im Ganzen vier verschiedene Wege, um auf den Typ eines Wertes oder einer Variablen Einfluß zu nehmen: Zunächst ist da einmal das Aussehen der Zahlen selbst, wenn sie in einem Programm oder in der direkten Betriebsweise des Computers als „Taschenrechner" verwendet werden und nicht etwa einer Variablen einen Wert zuweisen.

Dann faßt der Computer ganze Zahlen im bekannten Intervall zwischen −32768 und +32767 als ganzzahlig auf, bis siebenstellige Werte als Zahlen mit einfacher Genauigkeit und längere als 16-stellig und verarbeitet sie auch so. Ein weiteres Unterscheidungsmerkmal zwischen den 6- und 16-stelligen Zahlen ist bei der wissenschaftlichen Schreibweise der Buchstabe zwischen Grundzahl und Exponent: E für einfache, D für "doppelte" Genauigkeit – z. B. 2,5 E+09, 7D−03.

Sind Variable mit im Spiel, bestimmen sie in den meisten Fällen auch den Variablen-Typ. Dafür gibt es drei verschiedene Zeichen: % für ganze Zahlen, ! für kleinere und # für die größere Genauigkeit. Mit den beiden letzteren läßt sich auch die Behandlungsweise von Zahlen beeinflussen. Wie das bekannte $ für Zeichenreihen werden auch %, ! und # dem Variablennamen (bzw. der Zahl) nachgestellt.
Zusätzlich kann man aber auch bestimmte Variable von vornherein als ganzzahlig, einfach oder doppelt genau definieren. Dazu dienen die Anweisungen DEFINT, DEFSNG und DEFDBL. Mit einer Liste von

Buchstaben oder einem Bereich des Alphabets oder beidem versehen (z. B. DEFDBL A,C,F,V-Z), bestimmen sie die mit den jeweiligen Buchstaben beginnenden Variablen als dem betreffenden Typ zugehörig. Die kann dann ein nachgestelltes, einem anderen Typ zugehöriges Zeichen im Einzelfall ändern. Natürlich sollten diese Anweisungen, wenn man sie verwendet, tunlichst am Anfang des Programms stehen. Wird nämlich irgendeine Variable mitten im Programmablauf plötzlich auf diese Weise neu definiert, verliert sie ihren augenblicklichen Wert, und anders als einer neu definierten Matrix erfolgt keine Fehlermeldung, die auf den Fehler aufmerksam machen könnte. Für Zeichenreihen gibt es übrigens auch die entsprechende Anweisung – DEFSTR.

Die "Funktionen"(eigentlich nur funktionsähnliche Anweisungen) CINT(x), CSNG(x) und CDBL(x) liefern schließlich ganzzahlige, 6- bzw. 16-stellige Darstellungen irgendwelcher numerischer Ausdrücke. CINT(x) unterscheidet sich dabei äußerlich von der Funktion INT(x) nur dadurch, daß es auf den Intervall der ganzzahligen Variablen beschränkt ist und scheint damit wenig Sinn zu haben. Doch ist der Zweck von CINT(x) nicht nur die Umwandlung in eine ganze Zahl allein, sondern die Nutzung der speziellen Eigenschaft der ganzzahligen Darstellungsweise, nämlich hohe Rechengeschwindigkeit, für eine bestimmte, einzelne Rechenoperation.

Entsprechendes gilt auch für CSNG(x) und CDBL(x). Anders als bei festen Zahlen läßt sich das bei Variablen mit Hilfe der Typ-Zeichen nicht machen, da ja verschiedene Zeichen dieser Art in Zusammenhang mit dem sonst gleichen Variablen-Namen verschiedene Variablen bedeuten. A% ist eben nicht die ganzzahlige Version von A!, sondern ein ganz anderer Wert.

Wenn man ganz auf die Typdefinitionen, Variablenzeichen u. s. w. verzichtet (bei den Strings geht das natürlich nicht), arbeitet der Computer selbsttätig mit einfacher Zahlengenauigkeit. Dies ist wie gesagt erfahrungsgemäß in den allermeisten Fällen auch ausreichend, und man wird weder die Anweisungen und das Zeichen für die einfache noch für die doppelte Genauigkeit viel verwenden. Andererseits sollte man sich auf jeden Fall für bestimmte Aufgaben die größere Schnelligkeit der ganzzahligen Größen zunutze machen. Am besten und einfachsten läßt sich das machen, indem man einige Buchstaben am Anfang des

Programms mit dem DEFINT-Befehl reserviert. Damit werden dann die Zählaufgaben in Programmschleifen sowie ähnliche Funktionen ausgeführt, was merklich beschleunigend auf die Programmausführung wirkt und nebenbei auch Speicherraum spart.

Auf die Definition von String-Variablen kann man natürlich nur verzichten, wenn man in dem betreffenden Programm ganz ohne Zeichenreihen auskommt. Damit bedient man sich allerdings der äußerst eleganten Programmiertechniken, die gerade das Radio Shack Level II BASIC auf diesem Gebiet bietet. Das DEFSTR hat zwar keine funktionellen Vorteile gegenüber dem String-Zeichen $. In manchen Programmen, die besonders auf den geschickten Einsatz von Zeichenreihen abzielen, kann man sich aber das Schreiben einer ganzen Menge von $-Zeichen ersparen.

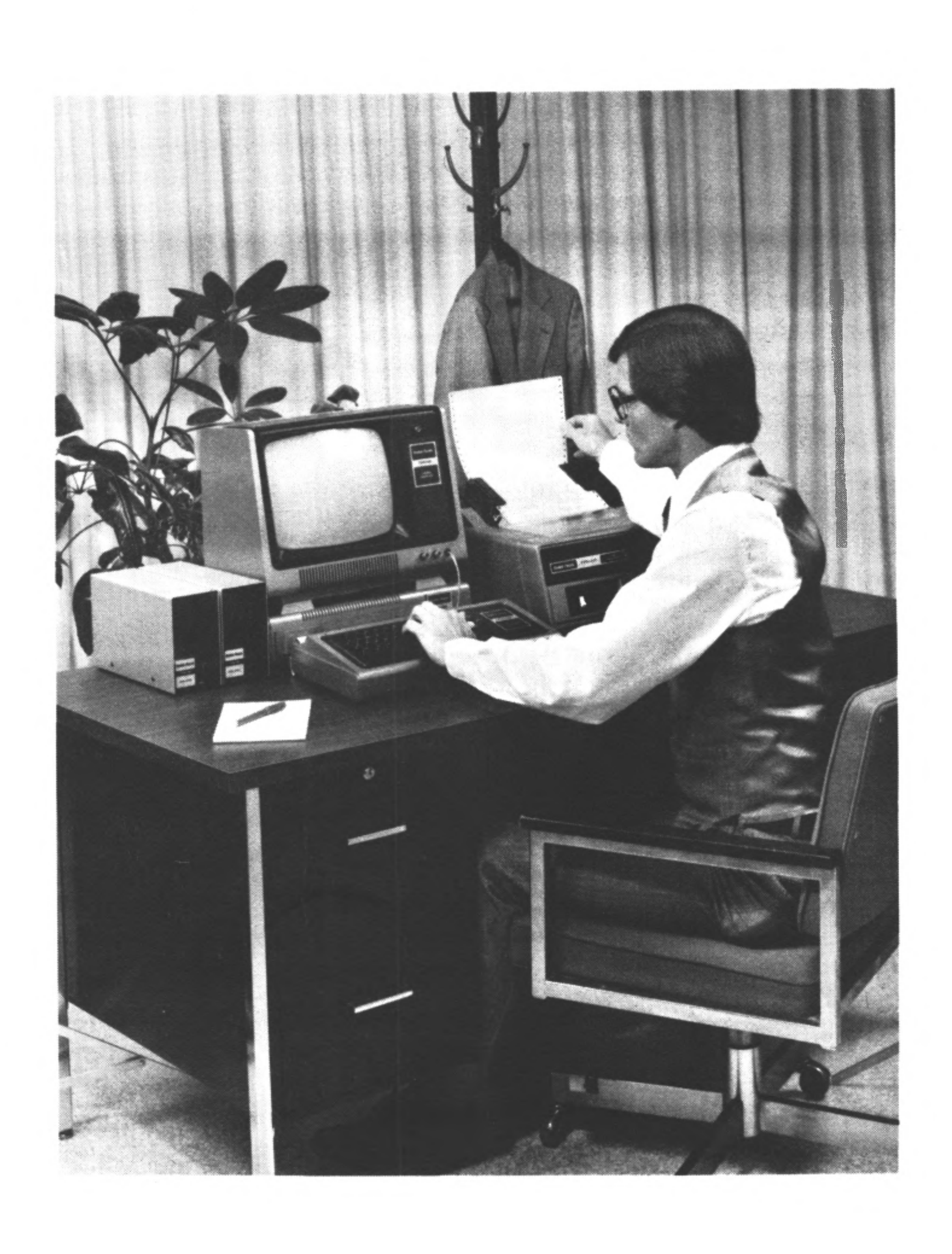
Die bei der gemischten Anwendung der verschiedenen numerischen Variablen-Typen gegebenen Fehlermöglichkeiten beruhen vor allem darauf, daß Variablen mit einfacher Genauigkeit nicht in Größen mit 16 Stellen umgewandelt werden dürfen. Dabei kommen nämlich Zahlen heraus, die in den Stellen, die die ursprüngliche Zahl hatte, zwar mit ihr übereinstimmt oder fast übereinstimmt. Die restlichen Stellen bis zur Gesamtanzahl von 16 werden aber nicht mit Nullen ausgefüllt (die dann bei der Anzeige unterdrückt würden), sondern mit irgendwelchen zufälligen Ziffern. Damit sieht die Zahl natürlich anders aus als vorher und ein Fehler entsteht. Daß dies so leicht geschehen kann, liegt wiederum daran, daß der Computer Werte ohne besondere Vorschriften als einfache Variable behandelt, und dies kann an Stellen der Fall sein, wo man fest überzeugt ist, es nur mit 16-stelligen Werten zu tun zu haben. Wer würde zum Beispiel so ohne weiteres darauf kommen, daß es bei dem Ausdruck A#=2/3 der Fall ist? Dennoch ist es so, weil zuerst die Rechnung "2 durch 3" durchgeführt wird, und zwar ohne besondere Vorschrift und daher in einfacher Genauigkeit. Dann erhält das doppelt genaue A# den in einfacher Genauigkeit gebildeten Wert zugewiesen, und das Unglück ist geschehen.
Selbst ein so unschuldig aussehender Ausdruck wie A#=3.2 führt (fast immer) zu einem Fehler! Der Grund dafür ist: Zuerst wandelt der Computer den Wert 3.2, der für sich allein ja keine besondere Genauigkeitsanweisung hat, in den entsprechenden Binärwert um, und zwar in sechsstelliger Genauigkeit. Dann folgt die – jetzt fehlerhafte

– Zuweisung zur doppelt genauen Variablen A#. Richtig ist diese Schreibweise: A# = 3.2#. Andererseits führt eine Zahleneingabe im Programmablauf in der Form INPUT A# natürlich auch dann zum richtigen Ergebnis, wenn man beim Eintippen das # wegläßt. Mit dem # Zeichen ist es ebenfalls in Ordnung. Wenn man allerdings das ! für die einfache Genauigkeit der eingegebenen Zahl anfügt, gibt es wieder den Fehler. So wird nämlich die interne Bildung des einfach genauen Wertes erzwungen, ehe die doppelte Genauigkeit zum Zuge kommt. Anders herum, das heißt bei Eingabe einer doppelten genauen Zahl auf Anforderung einer "normalen", (etwa indem man eine mehr als 7-stellige Zahl eingibt, das # verwendet oder die Form "xxx D xx" wählt), gibt es wiederum keinen Fehler.

Zu den ganzen Zahlen an dieser Stelle noch zwei Bemerkungen: Ein Ausdruck dieser Art: 45.3%, also ein Dezimalbruch mit angehängtem % für ganzzahlige Variable führt immer zu einem Syntax-Fehler, selbst wenn man es als Reaktion auf eine INPUT-Anweisung im Programm eintippt. Wenn man auf INPUT A% eine mehr als siebenstellige Zahl eingibt (im zugelassenen Bereich –32768 bis 32767) können manchmal seltsame Dinge geschehen, obwohl eigentlich immer automatisch abgerundet werden soll. 1.9999999 wird richtig zu 1, aber 1.99999999 zu 2. 32767.999 ergibt noch die größtmögliche Zahl 32767,32767. 9999 erzeugt aber bereits die Fehlermeldung OV ERROR für den Überlauf.

Etwas umständlich wird es, wenn eine 16-stellige Variable einen Wert in der Form eines Rechenausdrucks zugewiesen bekommen soll – in der Art wie oben ausgeführt. Dann muß nämlich jeder Dezimalbruch (d. h. jede Zahl mit "etwas hinter dem Komma") ein #-Zeichen bekommen, wenn die Zahl als Ganzes keine 16 Stellen hat. Bei manchen Brüchen kann es auch so gut gehen, aber man sollte sich darauf nicht verlassen. Ganze Zahlen innerhalb eines Rechenausdrucks werden auch ohne # richtig verarbeitet, aber nur, wenn man sich an bestimmte Regeln hält: Man muß dafür sorgen, daß die Berechnung mit doppelter Genauigkeit beginnt und nicht irgendwo von einfacher Genauigkeit zur doppelten überspringt.

Einerseits rechnet der Computer von links nach rechts. Daher muß eine der ersten beiden Zahlen einer Kette von Multiplikationen und



Werkbild von Fa. TANDY: TRS-80 mit Floppy-Disk und Drucker

Divisionen das # erhalten. Es sei denn, die erste Berechnung von links hat als Ergebnis eine ganze Zahl. Dann funktioniert die Sache auch mit dem # hinter der dritten Zahl, und so fort. Andererseits werden bei gemischten Ausdrücken ("Punkt-" und "Strichrechnung") die Multiplikationen/Divisionen enthaltenden Teile jeweils in der angeführten Weise mit dem # versehen. Der Computer führt sie nämlich jeweils getrennt als erstes aus (wie es ja auch den bekannten Rechenregeln entspricht), und dies muß natürlich jedesmal in der richtigen Weise, nämlich mit 16-stelliger Genauigkeit geschehen.

All dies muß auch beachtet werden, wenn man den TRS-80 als „Taschenrechner" gebraucht und dabei ohne Programm mit der größeren Genauigkeit rechnen will.

Man sieht also, die Anwendung der 16-stelligen Werte ist nicht ganz problemlos, und es kann sicherlich nicht schaden, sich die Verfahrensregeln mit ein wenig Geduld gründlich zu eigen zu machen, ehe man sich an entsprechende Programme wagt. Eventuell auf diese Weise in ein Programm eingebaute Fehler sind ja nicht ohne weiteres festzustellen und erzeugen auch keine Fehlermeldung.

Bei den anderen Möglichkeiten zum Übergang zwischen numerischen Größen unterschiedlicher Art gibt es glücklicherweise keine solchen Fußangeln. Bei einer Übertragung von einem 6- oder 16-stelligen Wert auf eine ganzzahlige Größe wird wie in anderen BASIC-Versionen ebenfalls üblich immer abgerundet. Übrigens haben alle Funktionen, die eigentlich nur mit ganzen Zahlen arbeiten, diese Eigenschaft gewissermaßen zum Teil mit eingebaut, so daß sie auch mit 6-stelligen Werten arbeiten können (nicht aber mit 16-stelligen). Die übliche „kaufmännische" 4/5-Rundung erreicht man übringens, indem man zu der jeweiligen Zahl 0.5 hinzuaddiert. Dadurch werden alle Werte über ...,5 über die nächsthöhere ganze Zahl hinausgeschoben und deshalb zu dieser abgewertet. So kommt dann die erwünschte Aufrundung heraus. (Ein Beispiel: A=3,9: INT(A)=3 aber INT(A+.5)=4, weil A+.5=4.4 ist)

Beim Übergang von 16- auf sechsstellige Zahlen soll eigentlich korrekt nach 4/5 gerundet werden. Doch ist hier auch ein wenig Vorsicht geboten: Ganz richtig funktioniert die Sache nicht. Aus der genaueren Zahl 0,44444444 wird zum Beispiel nicht etwa 0,444444, was richtig

wäre, sondern der Computer rundet falsch zu 0,444445 auf. Sicher lassen sich noch mehr solche Beispiele finden, wenn man sich an der Grenze zwischen Auf- und Abrundung bewegt. Wenn es in einem speziellen Fall darauf ankommt, sollte man dies vielleicht ausprobieren, ehe die entsprechende Befehlszeile in ein Programm eingebaut wird.

Ein- und Ausgabe

Bei der Eingabe von Daten im laufenden Programm bewegt sich der TRS-80 so ziemlich auf der Linie der übrigen Microcomputer. Mit dem INPUT-Befehl lassen sich Daten von der Tastatur übernehmen. Dabei ist wieder zwischen den verschiedenen Variablen-Typen zu unterscheiden. Da INPUT sich auch mit einer Liste von Zahlen und/oder Strings ausstatten läßt, berücksichtigt das Betriebssystem des Computers, ob die korrekte Datenmenge in der richtigen Reihenfolge eingegeben wurde – allerdings in seiner recht kargen Art, von der schon vorher die Rede war:

Wenn noch etwas fehlt, erscheinen zwei Fragezeichen. Fehler bei der Zahleneingabe beantwortet er mit einem simplen "REDO", was auf Deutsch "mach' es noch einmal" heißt. Wird zuviel eingegeben, d. h., ist die durch Kommas getrennte Datenliste zu lang, rafft er sich zu einem "EXTRA IGNORED" ("Restliches nicht beachtet") auf und übernimmt nur das, was er braucht.

Dies kann relativ leicht passieren, da an einem Komma für den Computer eine Zahl oder eine Zeichenkette zuende ist - im letzteren Fall, wenn sie nicht in Anführungszeichen steht. Setzt man also versehentlich nach deutscher Art ein Komma statt eines Dezimalpunktes, oder vergißt ein String, die Kommas (oder Doppelpunkte) enthalten, die Anführungszeichen, bekommt man das EXTRA IGNORED als Antwort. Am Ende einer Eingabeliste, oder wenn man die Zeichenreihen einzeln eintastet und dazwischen immer wieder ENTER drückt, kann man sich das zweite Anführungszeichen wie beim PRINT-Befehl schenken.

Recht praktisch ist es, daß man bei der Eingabe längerer Zahlen beliebige Zwischenräume zwischen den einzelnen Ziffern machen kann. So kann man sie zur besseren Übersichtlichkeit zum Beispiel in Dreiergruppen gliedern. Vorsicht ist wieder bei der Eingabe von Daten in der 16-Stellen-Form geboten. Wählt man dabei die wissenschaftliche

Notierung, muß unbedingt das die 16 Stellen kennzeichnende D zwischen Grundzahl und Exponent stehen (nicht das E!). Sonst kommt es zu der unzulässigen, fehlerbehafteten Übertragung eines Wertes von der sechs- zur 16-stelligen Form. Ansonsten ist diese Form der Eingabe besonders bei sehr großen sowie sehr kleinen Zahlen mit vielen Nullen vor oder hinter dem Komma zeitsparend und daher empfehlenswert. Den Computer stört es auch nicht, wenn man dabei von der Standardschreibweise (z. B. 4.57834E+02) abweicht und das Komma beliebig setzt, sowie das Vorzeichen und die Null des Exponenten fortläßt. Auch kann man bei einfachen Variablen ruhig das D statt dem E verwenden – der Computer wandelt dann eben einen 16-stelligen Wert in eine 6-Stellen-Größe um. Aber wie gesagt, anders herum entstehen Fehler.

INKEY$

Die Anweisung, mit der sich Daten ohne die ENTER-Taste (und damit auch ohne Programm-Unterbrechung) vom Tastenfeld holen lassen, heißt beim Radio Shack-BASIC INKEY$. Wie das $ am Ende schon andeutet, ist es der Form nach keine Anweisung, sondern eine String-Funktion. Entsprechend muß INKEY$ auch im Programm verwendet werden: Der Befehl, oder besser die Wertzuweisung, die man verwenden muß, heißt A$=INKEY$, und damit übernimmt das Programm ein Zeichen von der Tastatur und keinen Zahlenwert. So muß nicht zwischen Buchstaben und Zahlen unterschieden werden, aber zur Übernahme von Ziffern ist dafür noch die Funktion VAL(A$) einzusetzen (in der Form A=VAL(A$)).

Die Art, in der Information vom Tastenfeld in den Computer gelangt, hat für den INKEY$-Vorgang charakteristische Folgen, die man in manchen Fällen nicht außer Acht lassen kann: Zunächst erzeugt jedes Drücken einer Taste nur ein einmaliges Signal, so daß wiederholtes Abfragen mit INKEY$ erst beim jeweils nächsten Tastendruck zu einem Resultat führt. Durch einfaches Niederhalten einer Taste kann man keinen dauernden Vorgang auslösen. Doch außerdem übernimmt INKEY$ immer das zuletzt durch Tastendruck erzeugte Signal, auch wenn dies schon länger zurückliegt. Das kann zu Schwierigkeiten führen, wenn nach Auftauchen des INKEY$ erst ein Tastendruck abgewartet werden soll. Abhilfe schafft in diesem Fall ein zweites, kurz vorher im Programm (aber nicht in der Warteschleife des eigentli-

chen INKEY$) angeordnetes INKEY$, das die Tastatur gewissermaßen zunächst abräumt (und keine weitere Funktion im Programm hat).

Eine der besten Prüfsteine für die wahren Qualitäten eines Computers – und nicht nur eines Microcomputers, wohlgemerkt – sind die Möglichkeiten der Datenausgabe. Und hier kann der TRS-80 wahrhaft glänzen. Wenn sich dies alles auch bei der normalen Geräte-Konfiguration nur auf dem Bildschirm abspielt – was Radio Shack hier in Verbindung mit der 16-stelligen Genauigkeit mit seinem Level II-BASIC bietet, weist geradewegs zur ernsthaften kommerziellen Anwendung, die mit den Ausbaumöglichkeiten des Systems ins Auge gefaßt wird. Wenn dies die Möglichkeiten des privaten Anwenders im allgemeinen wohl auch übersteigen wird. Doch auch ihm stehen die vielfältigen Spielarten der Datenausgabe bereits für seinen Bildschirm zur Verfügung.

Die Standardausstattung der Microcomputer – PRINT mit dem Tabulator TAB(x) und eventuell auch der Funktion POS(x), die die momentane Position des Cursors auf der Zeile angibt – sind hier natürlich auch vorhanden, zusammen mit den üblichen Formatzeichen ; und , mit denen man anschließen läßt bzw. vier Spalten auf dem Bildschirm erzeugt . Wenn es dabei keine Unklarheiten gibt, kann man das Semikolon auch weglassen und die verschiedenen, zu druckenden Werte und Zeichenreihen im Programm unmittelbar zusammenschreiben.

Aber schon ungewöhnlicher ist das PRINT @, über das der TRS-80 verfügt. Mit einer Zahl zwischen 0 und 1023 versehen, läßt diese Anweisung das zu Druckende an dem betreffenden der 1024 Druckpositionen des Bildschirmes erscheinen (16 Zeilen zu je 64 Stellen). Damit kann man jeden Punkt des Schirms also direkt erreichen. Laut Anleitung gibt es bei dieser Anweisung anschließend den üblichen "Wagenrücklauf", d. h., der Cursor, der während der Programmausführung allerdings normalerweise nicht sichtbar ist, rückt auch in der letzten Zeile zum Anfang der nächsten Zeile vor und hebt dabei das ganze Bild um eine Zeile. Dies soll sich mit dem bekannten ; unterdrücken lassen. Leider gilt dies für die letzte Stelle jeder Zeile nicht, da der Cursor ja auf jeden Fall zur nächsten Stelle vorrückt, die in diesem Fall eben auf der nächsten Zeile liegt. Für die letzte Zeile bedeutet das aber, daß sich deren letzte Stelle mit PRINT@ nicht nutzen läßt, ohne das Bild hochrücken zu lassen.

Vollends ungewöhnlich für einen Microcomputer dieser Art ist schließlich die mit PRINT zusammen verwendbare Formatanweisung USING. Mit ihrer Hilfe lassen sich Zahlen mit beliebiger Stellenzahl vor und hinter dem Komma, mit vorangehender oder nachfolgender Bezeichnung, mit Vorzeichen vorne und hinten u. s. w. ausdrucken. Möglich ist es auch, die wissenschaftliche Schreibweise stets zu erzwingen, doch in der Praxis wird man wohl genau den umgekehrten Weg gehen und die Ausgabe von Zahlen mit Grundzahl und Exponent mit PRINT USING unterdrücken. Das gewünschte Format wird jeweils in einer Zeichenreihe angegeben, die sich praktischerweise auch als String-Variable im Programm verwenden läßt. Am besten wird man wohl daran tun, sich die Regeln für die Bildung dieser Zahlenreihen einmal vorzunehmen und selbst auszuprobieren, was hier alles möglich ist.

Dazu noch ein paar Hinweise, die vielleicht nicht mit genügender Deutlichkeit aus der Anleitung für den Computer hervorgehen: Zunächst zeigt sich gerade auch mit dem PRINT USING die amerikanische Herkunft der Maschine recht deutlich. Eine der Möglichkeiten besteht nämlich darin, ein Dollarzeichen vor der Zahl „schwimmen" zu lassen. Für uns wäre "DM" hier sicher angebrachter, damit geht es aber leider nicht. Durch ein irgendwo im Zahlenfeld vor dem Dezimalpunkt angeordnetes Komma läßt sich dieser Teil der Zahl in Dreiergruppen einteilen, die jedoch ebenfalls in englisch-amerikanischer Manier durch Kommas getrennt sind.

Verschiedene Formathinweise lassen sich in der Zeichenreihe kombinieren, doch muß das Vorzeichen dabei am Anfang oder Ende des Zahlenfeldes stehen. Wohlgemerkt: des Zahlenfeldes. Außerhalb können dann noch beliebige Namen, Bezeichnungen u. s. w. stehen, die dann entsprechend mit abgedruckt werden. Wird das Vorzeichen nicht extra spezifiziert, muß dafür gegebenenfalls (wenn negative Zahlen zu erwarten sind) eine Stelle im Zahlenfeld reserviert werden.

Das Zahlenfeld darf einschließlich Vorzeichen, Dezimalpunkt u. s. w. höchstens 24 Stellen lang sein, sonst erfolgt eine Fehlermeldung. Allerdings hat diese Einschränkung wenig praktische Bedeutung, da sich auch mit PRINT USING nur höchstens 16-stellige Zahlen in die normale Schreibweise bringen lassen. Längere werden in jedem Fall in wissenschaftlicher Notation gedruckt.

An dieser Stelle ist vielleicht auch eine Bemerkung zu den Möglichkeiten von Interesse, die beim TRS-80 zur schriftlichen Datenausgabe bestehen: Wie bekannt, läßt das Interface-Bauteil, mit dem sich der Computer erweitern läßt, auch den Anschluß eines Druckers zu. Dabei ergibt sich eine Erweiterung des BASICs um die beiden Befehle LLIST und LPRINT, die mit (fast) allen Spielarten der entsprechenden Anweisungen PRINT und LIST die Tätigkeit des Druckers kontrollieren. Nun sind die beiden Druckerbefehle in Wahrheit schon im normalen Level II-BASIC enthalten. Sofern sich also Drucker oder auch elektronisch gesteuerte Schreibmaschinen direkt an den Computer anschließen lassen – und dies ist im Prinzip tatsächlich möglich – kann er die Geräte ebenso gut auch ohne Erweiterungsbauteil steuern. Davon soll später noch die Rede sein.

Auch als Speicher für größere Datenmengen verwendet der TRS-80 bekanntlich einen ganz gewöhnlichen Cassettenrecorder. Und das geschieht in diesem Fall mit zwei ganz simplen Befehlen, nämlich PRINT# –1 und INPUT#–1. Formatierung in sogenannten Files, die gezieltes Ablegen und wieder Aufsuchen von Daten zulassen, ist in diesem System nicht vorgesehen. Es liegt nahe, daß man sich diesen Aufwand gespart hat, weil das Speichern und Zurückholen der Daten mit dem dafür nicht konstruierten Tonbandgerät eine derart zeitraubende Angelegenheit ist, daß der Benutzer zumindest bei größeren Datenmengen gezieltes Suchen und Ablegen bald aufgeben und sich auf das Bewegen zusammenhängender Datenblöcke beschränken wird.

Dazu noch zwei Anmerkungen, die man im Auge behalten sollte, um keine Enttäuschungen zu erleben: Zahlen und Zeichen werden durch den Computer in unabhängigen Abschnitten jeweils mit einem PRINT#–1-Befehl auf das Band geschrieben. Diese Blöcke dürfen nicht, wie in der Anleitung angegeben, bis 255 sondern nur 249 Stellen lang sein.

Was darüber hinausgeht, wird kurzerhand abgeschnitten und man wundert sich beim Wiedereinlesen, wo sie wohl geblieben sein können. Auch werden Zahlen so, wie sie auf dem Bildschirm erscheinen, Ziffer für Ziffer auf das Band geschrieben. Das bedeutet, die siebte bzw. 17. Ziffer, die ja intern vorhanden ist, aber aus Genauigkeitsgründen nicht

angezeigt wird, geht dabei verloren. Wenn man also eine Zahlenreihe addiert, dann speichert, wieder aufruft und wieder addiert, ergibt sich eine Differenz zur ersten Summe. Dieser Unterschied bleibt allerdings im allgemeinen auch bei der Summe auf die unsichtbare letzte Stelle beschränkt. Tests auf richtiges Speichern und Lesen von Daten, die mit einem Vergleich nach diesem Muster durchgeführt werden, scheitern also, und es sieht dabei so aus, als ob der Computer geheimnisvollerweise an völlig gleichen Zahlen Unterschiede entdeckt. Im Prinzip ist ein solcher Test allerdings auch nicht notwendig, denn der Computer kontrolliert bereits selbst richtiges Einlesen und meldet sich bei einem Fehler mit einer Fehleranzeige.

Um Bedienungsfehler zu vermeiden, empfiehlt sich allerdings eine andere Art Sicherung in den Speicher- und Lesevorgang einzubauen:
Man kann beim Speichern an den Anfang jedes Datenpakets einen Buchstaben oder eine Ziffer setzen, mit dessen Hilfe der Computer die Daten dann beim Lesen wieder erkennt und gleich bemerkt, wenn man zum Beispiel an der falschen Bandstelle begonnen hat.

Schließlich gibt es auch beim TRS-80 die Anweisung READ, die in speziellen DATA-Zeilen abgelegte Daten ins eigentliche Programm übernimmt, und auch das RESTORE, das die Wiederholung dieses Vorganges ermöglicht, fehlt nicht. Viele Programmierer verwenden diesen Mechanismus übrigens, um das zeitaufwendige Speichern der Informationen mit dem doch recht umständlich arbeitenden PRINT# —1/INPUT#—1-Verfahren zu umgehen. Nachteilig ist dabei, daß das Programm bei der Datenübernahme keine Hilfe leisten kann, da sie ja in diesem Fall außerhalb des Programmlaufs durch Schreiben von Programmzeilen mit der Anweisung DATA... und einer simplen Liste von Zahlen und Zeichenreihen geschieht. Doch man kommt in den Genuß des Vorteils, daß die Daten gleich mit dem Programm zusammen gespeichert und ausgelesen werden. Eine erheblich schnellere Prozedur.

Als zusätzliche Fehlerquelle ergibt sich dabei allerdings, daß bei jeder Veränderung der Daten auch das Programm neu gespeichert werden muß. Und dabei empfiehlt sich, wie erwähnt, schließlich auch der Vergleich des gespeicherten mit dem noch im Computer befindlichen Programm , was ja ebenfalls Zeit kostet.

ON ERROR GOTO

Außer den unbedingten und bedingten Sprungbefehlen, die allen vergleichbaren BASIC-Versionen gemeinsam sind, bietet das Radio Shack Level II BASIC eine weitere Art des Sprungs, ON ERROR GOTO. Bedingung, daß ein solcher Sprung ausgeführt wird, ist das Auftreten irgendeines Fehlers, den der Computer entdecken kann. Auf den ersten Blick mag das wie eine Programmierhilfe aussehen, in Wirklichkeit ist es jedoch ein äußerst elegantes und wirkungsvolles Mittel zum Organisieren eines Programmablaufes, das man mit wachsender Programmiererfahrung immer mehr schätzen lernt. An den Anfang eines Programmes gestellt, hält die Anweisung gewissermaßen ständig Wacht, ob irgendwo ein Fehler passiert, und veranlaßt daraufhin den Sprung zu der für diesen Fall vorgesehenen Zeile. Hier sind Anweisungen zu finden, die den Fehler in passender Weise korrigieren, und mit dem Befehl RESUME kann der normale Programmablauf mit einem Rücksprung wieder aufgenommen werden. Um die verschiedenen, möglicherweise auftauchenden Fehler für die jeweils angemessene Behandlung auseinanderhalten zu können, gibt es zusätzlich noch zwei Funktionen: ERL liefert die Nummer der Zeile, in der die Panne passierte, ERR die Art des Fehlers. Merkwürdigerweise muß man jedoch den Ausdruck ERR/2+1 bilden, um zu den bereits erwähnten Fehler-Codezahlen zu kommen. Beim RESUME gibt es mehrere Möglichkeiten – Rücksprung zur selben, zur jeweils nächsten oder zu einer eigens zu bestimmenden Zeile.

Die Möglichkeiten, diesen Mechanismus zur Gestaltung eines Programmes zu verwenden, sind fast unerschöpflich. Wie bereits in der Anleitung für den Computer ausgeführt, lassen sich zum Beispiel Fehler ausbügeln, die bei der Dateneingabe passieren können. In der Fehlerbehandlungsroutine wäre dann auch eine Anweisung zum Drucken einer Erklärung für den Benutzer enthalten, die auf den Fehler hinweist. Weiterhin braucht man sich bei der Anwendung um so manche Vorkehrung nicht mehr kümmern, die normalerweise für den Fall erforderlich sind, daß einer Funktion ein Zahlenwert außerhalb seines Definitionsbereichs zugeordnet wird. Einfachstes Beispiel hierfür sind die Graphik-Funktionen des TRS-80, nämlich SET(x,y), RESET(x,Y) und POINT(x,y). Von ihnen soll noch zu sprechen sein, doch hier mag genügen, daß die Koordinaten x und y Werte zwischen 0 und 127 bzw. zwischen 0 und 47 annehmen können. Soll etwa eine geometrische Figur gezeichnet werden, von der sich absehen läßt, daß sie über den

Rand des Bildschirms reicht, braucht man nichts weiter zu tun, als eine ON ERROR GOTO-Routine vorzusehen, die in der Zeile zur Fehlerbehandlung lediglich RESUME NEXT enthält. Dann überspringt der Computer die Zeile mit der graphischen Anweisung so lange, bis der Bildschirm wieder erreicht und der Graphik-Befehl wieder sinnvoll ist.

In dieser einfachen Form gilt die Anweisung natürlich für alle Fehler, nicht nur den, den man gerade im Auge hat. Sind noch andere Fehler zu erwarten, muß innerhalb der ON ERROR GOTO-Routine mit Hilfe von ERL und ERR spezifiziert werden.

Über das mehr oder weniger zwangsläufige Auftreten von derartigen Problemen hinaus, für die ja auch ohne ON ERROR GOTO irgendwie Vorsorge getroffen werden muß, läßt die Prozedur aber auch zum Durchorganisieren von Programmen einsetzen, in dem man von Fall zu Fall bewußt Fehler einbaut, um den ON ERROR GOTO-Sprung auszulösen. Dieses Vorgehen ist vor allem dann vorteilhaft, wenn die ON ERROR GOTO-Zeile sowie die zugehörige Fehlerberichtigung bereits aus anderen Gründen im Programm erforderlich ist.

Ist beispielsweise ein Programm in mehrere Teile gegliedert, die sich von einem Auswahl-Programmteil (sogenanntes Menü) aufrufen lassen und jeweils zur Eingabe von Daten unterschiedlicher Art dienen, läßt sich in einfachster Weise der Rücksprung zum Menü bewerkstelligen, das man ja immer wieder aufrufen muß, um zu einem anderen Teil des Programms zu gelangen. Man läßt jeden dieser Programmteile mit folgender Zeile beginnen:

```
xxxx I=0: INPUT I: J=1/I
```

Dies erzeugt einen /O-Fehler, wenn kein Wert für I eingegeben, sondern nur ENTER gedrückt wird. Steht dann in der von dem ON ERROR GOTO-Befehl angegebenen Zeile folgendes:

```
xxxx IF ERR/2+1=11    RESUME yyyy
```

dann landet man immer wieder beim Menü, das mit der Zeile yyyy beginnen muß. Sinnvoll ist diese Methode natürlich nur, wenn das INPUT I auf jeden Fall gebraucht wird, sonst wäre der Aufwand dafür unnötig groß.

Zur Fehlersuche beim Programmieren läßt sich das Verfahren offenbar nicht verwenden, denn es verhindert ja sogar die Anzeige von Fehlern. Entsprechend gibt es auch die Anweisung ON ERROR GOTO 0, die, innerhalb der Routine (vorübergehend) eingefügt, die Fehlermeldungen wie gewohnt arbeiten läßt, die Prozedur also außer Kraft setzt.

Den zur Fehlerbehandlung dienenden Programmteil, der mit dem ON ERROR GOTO-Befehl erreicht wird, stellt man günstigerweise an das Ende des Programms. Im Zuge des normalen Programmablaufs soll er ja nicht erreicht werden, und dazu müßte man sonst immer Sprünge über die entsprechenden Zeilen hinweg vorsehen. Wichtig ist auch, daß eine eventuell für das Programm erforderliche CLEAR-Anweisung immer vor dem ON ERROR GOTO steht. Ebenso wie zum Beispiel die Matrix-Dimensionierung mit DIM hebt CLEAR das ON ERROR GOTO wieder auf, so daß es, vorangestellt, wirkungslos bleibt.

Die Graphik des TRS-80

Auf dem Gebiet der graphischen Darstellungen auf dem Bildschirm hat sich Radio Shack, wie man wohl sagen muß, große Mühe gegeben, einen möglichst günstigen Kompromiß zwischen drei widerstrebenden Elementen zustande zu bringen. Private Benutzer legen Wert auf möglichst reizvolle Gestaltungsmöglichkeiten, doch dem setzt gerade der Kaufpreis des Systems, der auf diesen Personenkreis zugeschnitten werden soll, Grenzen. Schließlich bringen erwünschte kommerzielle Einsatzmöglichkeiten andere Anforderungen an Betriebssystem und BASIC mit sich, als gerade besonders gute Graphik auf dem Bildschirm.

Allerdings kann sich durchaus sehen lassen, was bei diesem Kompromiß herausgekommen ist. Jede der 1024 Druckpositionen des Bildschirms ist für graphische Zwecke in sechs Felder eingeteilt, und man erhält einen echten graphischen Bildschirm mit dem allerdings etwas groben Raster von insgesamt 6144 Bildpunkten. Diese lassen sich — und das ist der Hauptvorteil dieses Systems — nun mit den drei Befehlen SET(x,y), RESET(x,y) und POINT (x,y) direkt ansprechen und abfragen. Das Resultat sind außerordentlich einfache Programme zur Darstellung beliebiger Bilder — in dem wie gesagt etwas groben Punktraster. Das Zeichnen von Kurven mathematischer Funktionen wird auf diese Weise eigentlich überhaupt erst möglich, daher kann man in den Grenzen eines solchen Microcomputer-Systems von einer echten Bildschirmgraphik sprechen.

Der Nachteil dieser Methode ist seine vergleichsweise langsame Ausführung. Auch erfordern komplizierte Bilder, die nicht gerade mathematische Funktion oder etwas ähnliches repräsentieren, doch einen recht hohen Programmieraufwand. Doch gibt es für solche Fälle, oder wenn es schneller gehen soll (zum Beispiel bei einem Computerspiel), noch zwei weitere Möglichkeiten, zu graphischen Darstellungen zu kommen. Am schnellsten geht es mit den Zahlen des graphischen Codes (129 bis 191) die man, wie bereits ausgeführt, mit den beiden Funktionen CHR$(x) und STRING$(n,x) zu Zeichenketten zusammenstellen kann. Da jedoch immer die sechs Felder einer Druckposition auf einmal erfaßt werden, ist das Zusammenfügen der richtigen Zeichen zusammen mit den Codezahlen für die unter Umständen auch erforderlichen Cursor-Kontrollfunktionen (Cursor nach rechts, nach links u. s. w.) wahrlich keine ganz einfache Sache. Empfehlenswert ist es, beim Programmieren erst einmal folgende Zeile einzutippen, die man

dann wieder löschen kann, wenn das Programm fertig ist:

```
30000 N=128: FOR X=1 TO 7: FOR Y=1 TO 9: N=N+1: PRINT N
      CHR$(N) " " ; :NEXT Y: PRINT: PRINT: NEXT X
```

Durch ein GOTO 30000 kann man diese Zeile dann immer wieder aufrufen, wenn man unter den 63 verschiedenen graphischen Figuren die für den jeweiligen Zweck geeigneten heraussuchen will. Die möglichen Punktkombinationen werden schön ordentlich auf dem Bildschirm zusammen mit den zugehörigen Codezahlen abgebildet.

Ein Mittelding zwischen diesen beiden Methoden zur Programmierung von Graphiken stellt das dritte Verfahren dar, das mit dem Befehl POKE m,n arbeitet. Damit kann man bekanntlich eine Speicherstelle des Computers direkt mit einem Byte Information belegen, und zwar in Form einer Dezimalzahl zwischen 0 und 255. Nimmt man sich den Bildschirmspeicher des TRS-80 vor (Adressen 15360 – 16383) und verwendet die Zahlen des Graphik-Codes, kann man jedes Zeichen direkt an jede Stelle des Bildschirms bringen. Das geht schneller als mit SET(x,y), da immer sechs Bildpunkte zugleich erfaßt werden, aber nicht so schnell wie das Verfahren mit den Zeichenketten, da für jede Druckposition ein neuer POKE-Befehl erforderlich ist. Dafür muß man sich aber auch nicht mit dem Zusammenstellen und richtigen Ausdrucken der Zeichenketten herumschlagen.

Es wird wohl immer auf den Einzelfall ankommen, zu welcher der drei Methoden man greift, um Graphik zu programmieren. Mit wachsender Erfahrung lernt man auch immer besser unterscheiden, welches Verfahren der jeweiligen Aufgabe am besten angepaßt ist.

Mathematik und Logik
Auch auf diesem Gebiet wartet das Radio Shack Level II BASIC mit einigen Besonderheiten auf, über die durchaus nicht alle Microcomputer verfügen. Wie bereits ausgeführt, führt die Tatsache, daß es Variable mit sechs- und 16-stelliger Genauigkeit gibt, zu zusätzlichen Bestimmungs- und Umwandlungsanweisungen. Daneben sind die üblichen Rechenzeichen und Funktionen vorhanden, wobei wie gesagt zu beachten ist, daß (ohne die erwähnten Unterprogramme in Maschinensprache) die Funktionen nur in sechsstelliger Genauigkeit ausgeführt werden. 16-stellige Zahlen wandelt der Computer vorher entsprechend um. Klammern können in praktisch unbegrenzter Zahl ineinander verschachtelt werden — auch eine Eigenschaft, die nicht jeder Microcomputer bietet (der Autor hat seine Versuche bei 40 ineinandergeschachtelten Klammern aufgegeben!).

Bei den Rechen- und Logikzeichen wird folgende Reihenfolge der Bearbeitung durch den Computer angegeben:

↑ (potenzieren)
— (Vorzeichen wechseln)
* und / (multiplizieren und dividieren)
+ und — (addieren und subtrahieren)
=, 〉, 〈, 〉=, 〈=, 〈 〉 (Relationen)
NOT (logische Negation)
AND (logisches UND)
OR (logisches ODER)

Gleichrangige Operationen werden von links nach rechts durchgeführt. Bekanntlich sind ja die Klammern dazu da, eine andere Reihenfolge der Erledigung zu erzwingen. Wie die Erfahrung zeigt, machen diese Regeln beim Aufbau komplizierterer Ausdrücke manchmal mehr Schwierigkeiten als erwartet. Besonders bei längeren logischen Ausdrücken, die mit den Größenrelationen sowie AND, OR und NOT aufgebaut werden und in der Regel in bedingten Anweisungen (IF... THEN) stehen, muß man sich vorsehen. Ganz verzwickt kann es dann werden, wenn man die Möglichkeiten nutzt, IF-THEN-Klauseln auch noch ineinander zu verschachteln.

Zum Glück kann man hier jedoch mit Hilfe einer besonderen Eigenschaft des Level II-BASICs eine wesentliche Vereinfachung vornehmen. Eine zutreffende Bedingung drückt sich hier nämlich als die Zahl −1 aus, eine nicht zutreffende als 0, d. h., statt IF A = B THEN GOTO 100 kann man auch schreiben: IF (A = B) = −1 THEN GOTO 100.

In diesem Fall wäre das natürlich umständlicher, aber das muß nicht immer so sein. Der Ausdruck A=3 AND B=2 AND C=1 wird zum Beispiel zu (A=3) +(B=2) +(C=1) =− 3. Und wenn man schreibt IF (A=3) +(B=2)+(C=1)=−2, bedeutet das, jeweils zwei der drei Relationen müssen stimmen, damit die Bedingung erfüllt ist. Auf normale Weise ließe sich das nur so ausdrücken: IF A=3 AND B=2 AND C < > 1 OR A <> 3 AND B=2 AND C=1 OR A=3 AND B <> 2 AND C=1 − also wesentlich umständlicher. Und mit der Zahl der einzelnen Teilbedingungen wächst der Unterschied steil an.

Andererseits ist es aber auch möglich, folgenden Ausdruck zu bilden: IF (A=3)+(B=2) THEN GOTO 100. Man muß sich dabei wieder vorstellen, daß der Computer für jede wahre Aussage ein "−1" bildet und für jede falsche eine "0". Kommt dann bei einem derartigen Ausdruck, der natürlich auch aus noch mehr Gliedern bestehen kann, insgesamt nicht eine Null heraus, ist der ganze Ausdruck für den Computer „wahr", und er führt den an die Bedingung geknüpften Befehl aus, in diesem Fall also den Sprung nach Zeile 100. Man sieht also: (A=3) +(B=2) ist gleichbedeutend mit A=3 OR B=2. Entsprechendes gilt auch für das Rechenzeichen " * ". Auch hier prüft der Computer, ob bei einem zusammengesetzten Ausdruck eine Null herauskommt oder nicht und handelt dementsprechend − etwa bei dem Ausdruck (A=3)*(B=2). Da aber beim Multiplizieren immer Null herauskommt, wenn ein Faktor Null ist, müssen alle Teilbedingungen stimmen, damit der Computer die Bedingung als Ganzes als wahr einstuft. So entspricht das Zeichen *, logisch angewandt, dem AND.

Wohlgemerkt: Bei der Kalkulation ist in beiden Fällen nur maßgeblich, ob das Resultat Null ist oder nicht. Speziell −1 für das logische „wahr" wird für eine richtige Gesamtbedingung hier nicht verlangt. Wenn man auf diese Bedingung Wert legt, die bei der "+"-Operation einem „ausschließenden ODER" (exclusive OR) entspricht (nur eine einzige Teil-

bedingung darf richtig sein und nicht etwa mehrere oder alle), muß man zu der Konstruktion (...)+(...)+(...)+ ... +(...)= —1 greifen.

Man kann natürlich auch noch ganz andere logische Ausdrücke bilden, wenn man weitere Rechenzeichen einsetzt, z. B. ODER /, aber dann wird es wirklich schwierig, sämtliche möglichen Fälle zu durchdenken, so daß es zu unerwarteten Ergebnissen kommt. Mit den genannten Möglichkeiten sind andererseits aber auch alle normalerweise vorkommenden Fälle zu lösen.

Eine Anmerkung: Zwar läßt es das Radio Shack Level II BASIC zu, das THEN wegzulassen (bzw. das GOTO nach THEN oder ELSE), wenn dies zu keinen Unklarheiten führt. In den Beispielen steht es der Deutlichkeit wegen trotzdem da.

Überhaupt ist es eine ganz gute Übung, zumindest in der ersten Zeit, beim Programmieren erst einmal alle THENs mitzuschreiben und erst nach Fertigstellung des Programms die überflüssigen THENs und GOTOs wieder herauszunehmen, wenn man sicher ist, daß das Programm läuft. Manche Programmierer lassen auch absichtlich THENs und GOTOs im Programm stehen, die nicht erforderlich sind. Dies erleichtert die Lesbarkeit des Programms für eventuelle spätere Veränderungen.

Auch der umgekehrte Weg ist übrigens möglich: Man kann die logischen Operationen an Zahlen ausführen (ganze Zahlen von —32768 bis + 32767). Dieser auf den ersten Anblick verblüffende Vorgang beruht darauf, daß aus jeweils zwei Zahlen in ihrer binären Darstellung Bit für Bit unter Zugrundelegung der jeweiligen logischen Vorschrift eine dritte gebildet wird. Wie die Anleitung für den TRS-80 ganz richtig anmerkt, gibt es in der Geräte-Konfiguration des Computers ohne den Interface-Bauteil keine rechte Verwendung für diesen Vorgang. Den einen oder anderen speziellen Anwendungsfall wird man aber sicher finden.

Besonderes
Eine Besonderheit des TRS-80 ist zunächst vor allem auch die schon erwähnte breitere Schreibweise, bei der statt 64 nur 32 alphanumerische oder graphische Zeichen eine Zeile des Bildschirms ausfüllen. Dieser im Programmablauf (normalerweise) durch CHR$(23), in der direkten Betriebsart durch "SHIFT →" ausgelöste Vorgang läßt bei der Bildschirmausgabe eine recht reizvolle und für manche Zwecke nützliche Alternative zu. Bei der Programmierung sind aber spezielle Regeln zu beachten, deren Ursache in der Arbeitsweise des Computers liegen. Schreibt man irgendetwas auf den Bildschirm und schaltet dann mit „SHIFT→" auf die breite Anzeige um, werden die Wörter und Zahlen verstümmelt. Was danach geschrieben wird, erscheint richtig und doppelt so breit. Wie kommt das zustande? Bei der breiten Schreibweise fragt der Computer nur jede zweite Speicherstelle des Bildschirmspeichers ab und bringt ihren Inhalt auf den Schirm. Was in den anderen Stellen gespeichert ist, geht also verloren, und die Wörter, Zahlen usw. erscheinen verstümmelt. Gleichzeitig wird aber durch den PRINT-Vorgang auch nur in diese kleinere Zahl von Speicherstellen hineingeschrieben, so daß damit alles wieder seine Ordnung hat.

Zurückschalten auf normale Schrift ist nur durch die CLS-Anweisung bzw. durch die CLEAR-Taste möglich. Dadurch wird die Anzeige gleichzeitig gelöscht, und Hin- und Herschalten zwischen den beiden Schrifttypen ist deshalb nicht möglich, zumindest nicht mit diesem Verfahren.

Nun gibt es ja noch zwei weitere Methoden, etwas auf den Bildschirm zu bringen, nämlich mit POKE m,n und SET(x,y) zusammen mit RESET und POINT. Da diese Befehle Speicherstellen des Bildschirmspeichers direkt ansprechen, nimmt der Computer beim Umschalten der Typengröße keine Rücksicht darauf. Bei POKE darf man also nur noch die verwendeten Stellen (sie haben gerade Adressennummern) ansprechen.
Das gilt übrigens auch für PRINT @, das ja auch an jede Druckposition einzeln zu richten ist. Bei SET, RESET und POINT ist es ähnlich, nur sind die Druckpositionen in diesem Fall in der waagerechten Richtung, auf die es hier ankommt, zweigeteilt. Folglich sehen die „erlaubten" waagerechten Koordinatenzahlen jetzt so aus: 0,1, 4,5, 8,9, ... 124,125. Bei den senkrechten Koordinaten ändert sich nichts.

Mit INP(n) und OUT m,n verfügt der TRS-80 auch über die Möglichkeit, Signale von der „Außenwelt" in Form von beliebigen angeschlossenen Geräten zum Empfangen und auch dorthin zu senden. Kaum interessant, könnte man meinen, da sich ohne die Zusatz-Einheit Expansion Interface keine Geräte anschließen lassen. Doch ganz so ist es nicht. Zunächst gibt es auch einige wenige, aber dafür sehr interessante interne Vorgänge, die sich mit OUT steuern lassen. Und dann existieren auch bereits Zusatzgeräte, die sich direkt an den Grundbaustein anschließen lassen, und von denen später noch die Rede sein soll.

Doch zunächst zu den internen Funktionssteuerungen: Von den 256 Adressen, die man theoretisch mit dem Ausgabebefehl OUT ansprechen kann (0 bis 255) und die mit der ersten Zahl des der Anweisung angefügten Zahlenpaares genannt wird, ist die letzte für interne Vorgänge reserviert. Die Anweisungen OUT 255,8 bis OUT 255,11, OUT 255,24 bis OUT 255,27, OUT 255,40 bis OUT 255,43 u. s. w. schalten ebenfalls den Bildschirm auf breitere Anzeige um! Rückschalten kann mit OUT 255,0 bis OUT 255,3, OUT 255,16 bis OUT 255,19 u. s. w. geschehen. In diesem Fall paßt sich der PRINT-Vorgang nicht an, so daß es immer verstümmelte Wörter und Zahlen gibt. Aber beim Rückschalten wird der Bildschirm nicht gelöscht, so daß man zum Beispiel einen Schriftzug immer zwischen schmaler und breiter Schreibweise hin- und herspringen lassen kann, etwa folgendermaßen:

```
10 CLS
20 PRINT @ 530, "A C H T U N G ! ! !"
30 OUT 255, 8
40 FOR X=1 TO 200: NEXT
50 OUT 255,0
60 FOR X=1 TO 200: NEXT
70 GOTO 30
```

Wichtig sind dabei die gerade Zahl beim PRINT @ sowie die Zwischenräume in dem Wort "A C H T U N G ! ! !". Die „eigentlichen" Umschalt-Anweisungen setzen die OUT-Befehle außer Kraft, und sie funktionieren auch nur innerhalb eines Programms, im Gegensatz zu CHR$(23), dessen Wirkung auch nach einer Programmunterbrechung anhält.

Eine zweite überaus wertvolle Funktion des OUT 255,n ist die Kontrolle des Bandlaufs des Cassettenrecorders. OUT 255,4 bis OUT 255,7 schaltet ihn an, OUT 255,12 bis OUT 255,15 wieder aus. Wie oben gibt es auch hier wieder größere Zahlen, die denselben Effekt haben. Eine naheliegende Anwendung dieses Vorgangs ist zunächst einmal in Verbindung mit dem Speichern von Daten auf einer Tonbandcassette denkbar: Wenn man handelsübliche Bänder verwendet, muß man darauf achten, daß das Band bis an das Ende des Vorlaufbandes vorgelaufen ist, ehe der Speicherbefehl kommt, da sonst Daten verloren gehen. Dies kann der Befehl OUT 255,4 (Band anschalten) in Verbindung mit einer Warteschleife übernehmen:

```
100 OUT 255,4: FOR X=1 TO 5000: NEXT
```

Wenn X eine sechsstellige Variable ist (keine ganzzahlige), müßte die Wartezeit auf jeden Fall ausreichen, um das Band vom Anfang bis zum Beginn der Magnetschicht vorlaufen zu lassen. Wohlgemerkt – anschließend muß gleich PRINT# –1 kommen, da das Band nach OUT 255,4 immer weiter läuft. Soll das Band zunächst wieder anhalten, ist nach der Warteschleife ein zusätzliches OUT 255,12 erforderlich.

Mit diesem Mechanismus ist es darüberhinaus auch möglich, Programme aufzubauen, die Bildschirmanzeigen mit gesprochenen Kommentaren verbinden. Dazu braucht man nichts weiter zu tun, als die Kommentare (oder natürlich auch Musik, wenn man dies wünscht) auf eine Cassette aufzunehmen. Wenn das Programm startbereit ist, legt man die Cassette in das Cassettengerät des Computers, schaltet auf „Wiedergabe" und zieht den schwarzen, zur Ohrhörer-Buchse des Recorders führenden Stecker heraus. An passender Stelle im Programm muß jeweils ein OUT 255,4, eine Warteschleife und anschließend ein OUT 255,12 stehen. Dann wird der Recorder in Gang gesetzt und angehalten, und man hört nacheinander die einzelnen Kommentare, die der Recorder nun hörbar abspielt.

Die Länge der einzelnen Warteschleifen zwischen den beiden OUT-Befehlen muß man dabei natürlich dem jeweiligen Kommentar anpassen.

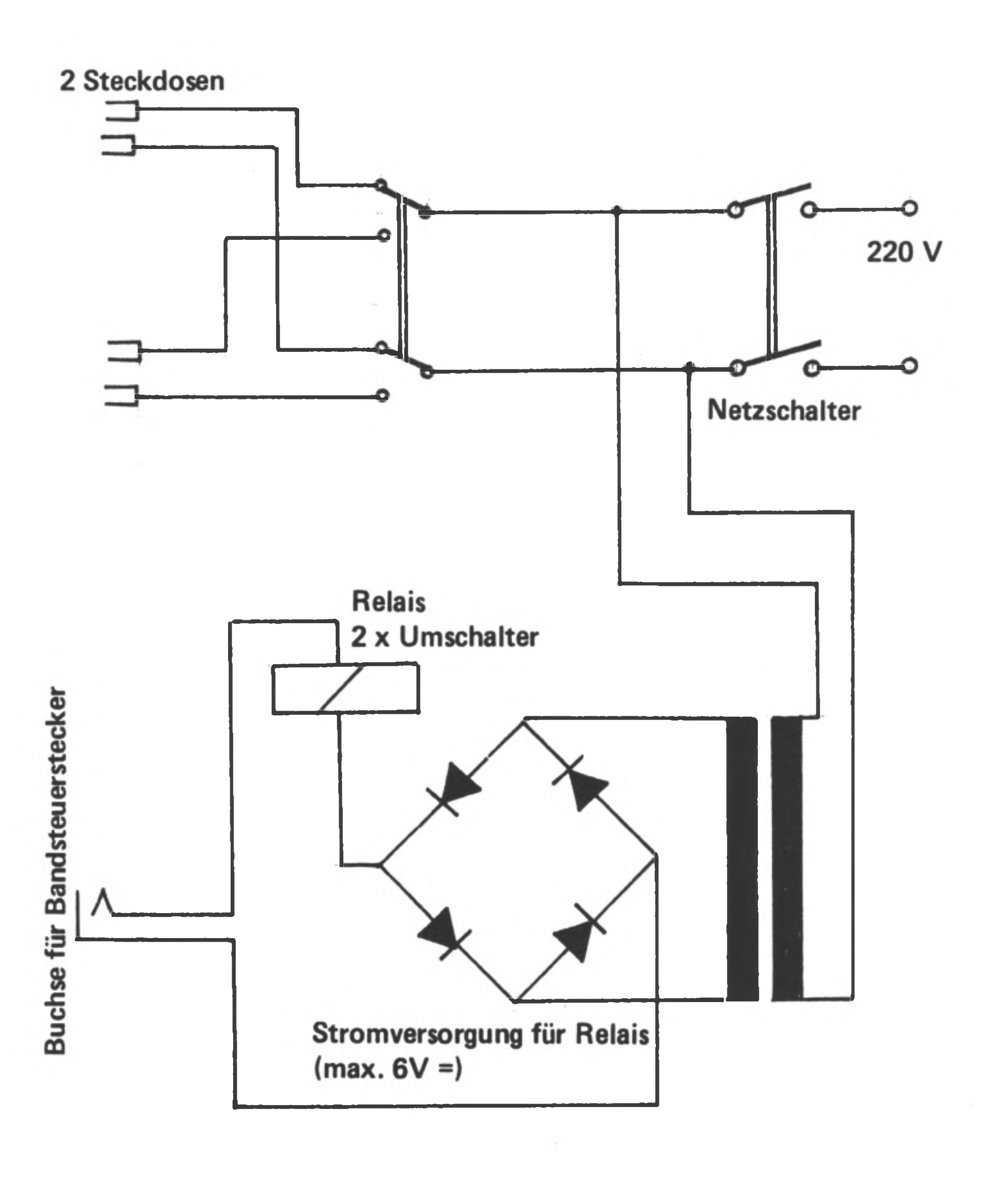
Im Grunde ist es nur ein kleines Relais, welches im Computer an- und ausgeschaltet wird, um das Tonbandgerät zu starten und zu stoppen. Diese Tatsache kann man ausnutzen, um den TRS-80 auch ein beliebiges anderes Gerät steuern zu lassen. Für das Relais gibt der Hersteller ein maximales Schaltvermögen von 0,5 A bei 6 V = (Gleichstrom) an. Um ein mit Netzspannung betriebenes Gerät vom Computer an- und ausschalten zu lassen, könnte man ein zweites Relais vom Relais des Computers steuern lassen. Diese Schaltung (Skizze) ließe sich, in einem geeigneten Gehäuse untergebracht, dazu verwenden. Vorgesehen sind zwei Netzsteckdosen, die, vom Computer kontrolliert, abwechselnd unter Strom stehen. Der Anschluß an den TRS-80 erfolgt über eine Buchse, die für den kleineren grauen Stecker des Verbindungskabels des Computer-Cassettengerät passend sein muß. Zum Steuern beliebiger Geräte muß dann nur noch dieser Stecker aus dem Recorder in das Kontrollgerät umgesteckt werden.

Und noch eine weitere Möglichkeit bietet das OUT 255,n: Bei jeder neuen Zahl n, die sich von der des letzten Befehls OUT 255,n unterscheidet, schickt der Computer auch einen Impuls in die Datenübertragungsleitung zum Cassettenrecorder. Günstigerweise sind nun, wie erwähnt, die Anschlüsse des Computers ganz gewöhnliche fünfpolige Diodenbuchsen nach DIN.

Und darüber hinaus stimmt auch noch die Lage des Ausgangsanschlusses mit dem entsprechenden Anschluß zwischen einem Tonbandgerät und einem Radio überein. (Ein Kanal eines Stereoanschlusses; der andere ist – allerdings erdfrei – mit der Fernsteuerung für den Cassettenrecorder belegt). Man braucht also nur ein fünfpoliges Verbindungskabel zu nehmen und den Computer an einen Radio oder einen Verstärker anzuschließen – und schon kann der TRS-80 selbst Töne hervorbringen. Folgendes kleines Programm genügt bereits dazu:

```
10 OUT 255,1
20 OUT 255,0
30 GOTO 10
```

Was man dann hört, wenn das Radio angeschlossen ist und das Programm läuft, ist ein ziemlich tiefer Ton – der Computer kann wegen



Schaltplan für die Steuerung mit dem OUT-Befehl

des vergleichsweise langsam arbeitenden BASICs die endlose Schleife, die das Programm darstellt, eben nicht allzuschnell durchlaufen. Und jeder weitere Zusatz, etwa um verschiedene Töne zu erzeugen, macht die Sache noch langsamer. Melodien wird man also kaum erzeugen können, eher verschiedene Geräusche. Ein Beispiel:

```
10 FOR X=1 TO 30: FOR Y=1 TO X
20 OUT 255,0: OUT 255,1
30 NEXT Y,X: GOTO 10
```

Statt gerade 0 und 1 kann man im Prinzip auch zwei beliebige andere Zahlen zwischen 0 und 255 an das Ende der beiden OUT-Anweisungen stellen. Sie sollten aber keine unerwünschten, anderen Wirkungen haben. Die Zahlen 0 bis 4 bedeuten hier ja „breite Schrift zurückschalten", was ja in diesem Zusammenhang nicht weiter stört.

Der TRS-80 könnte mit dieser Methode tatsächlich Musik erzeugen, wenn es gelingt, die Schleife so schnell zu machen, daß auch hohe Töne dabei herauskommen. Dies ist in der BASIC-Sprache zwar nicht möglich, wohl aber, wenn man den Computer mit der Maschinensprache dasselbe tun läßt.

Ein solches Programm ist weiter hinten im Buch vorhanden – im Kapitel über das Programmieren in Maschinensprache.

Mit den Befehlen PEEK und POKE, mit denen sich einzeln Informationseinheiten (Bytes) in individuellen Speicherstellen abfragen und unterbringen lassen ist, wie bereits bekannt, auch der TRS-80 ausgerüstet. Darüber hinaus gibt es aber noch eine ungewöhnliche Anweisung, nämlich VARPTR(x). Sie gibt an, an welcher Stelle im Speicher des Computers sich die Variable x befindet. Man mag vielleicht meinen, dies habe wenig Sinn – Zweck einer höheren Programmiersprache wie BASIC ist es ja eigentlich, gerade den Benutzer von derartigen Informationen unabhängig zu machen. Doch es gibt Fälle, in denen mit Hilfe des Speicherorts von Variablen spezielle Programmieraufgaben leichter zu lösen sind. Über VARPTR kommt man nämlich unmittelbar an die binäre Form der Zahlen heran, in der sie vom Computer gespei-

chert sind. Das kann zum Beispiel praktisch sein, wenn man den Wert einer Variablen an irgendein angeschlossenes Zusatzgerät schicken will. Dies muß ja mit dem Befehl OUT Byte für Byte geschehen, und mit VARPTR hat man diese Informationsabschnitte gleich richtig zur Hand und muß sie nicht erst aus der Dezimalzahl zurückrechnen, die der Computer beim Aufrufen einer Variablen normalerweise automatisch herstellt. Andersherum gilt natürlich das Gleiche: Eine in Byte-Darstellung übernommene Zahl kann direkt und ohne Umwege an seinen Speicherplatz befördert werden. Folgendes Programm zeigt mit Hilfe von VARPTR die (in der Anleitung erläuterte) Byte-Darstellung von eingegebenen 16-stelligen Zahlen:

```
10  INPUT A#
20  PRINT: FOR I=0 TO 7
30  PRINT PEEK (VARPTR(A#)+7–I)
40  NEXT: GOTO 10
```

Die größtmögliche positive Zahl sollte dabei so aussehen: 255 127 255 255 255 255 255 255, die größte negative Zahl 255 255 255 255 255 255 255 255. In der Praxis stimmt das nicht, da die letzte (siebzehnte) Ziffer der Zahlen ja nicht angezeigt wird, aber intern doch vorhanden ist und sich in der Byte-Darstellung niederschlägt. Man stellt aber fest, daß die in der Anleitung genannten größtmöglichen Zahlen nicht richtig sind: Nicht ± 1.701411834544556D+38 stimmt, sondern ± 1.701411834604692D+38.

Will man sich sechsstellige oder ganzzahlige Werte in Byte-Darstellung anschauen, muß man das Programm so verändern, daß nicht jeweils 8, sondern 4 bzw. 2 Byte angezeigt werden, und die Schreibweise der Variablen A# in A oder A% umändern.

Nachahmen der INPUT-Anweisung mit INKEY$

Die INPUT-Anweisung im Radio Shack Level II BASIC ist, wie überhaupt in der BASIC-Sprache, ein sehr praktisches Instrument für die Dateneingabe bei laufendem Programm. Doch gerade ihre wichtigste Funktion, nämlich das Programm anzuhalten, bis eine Zahl oder ein String oder mehrere davon eingegeben sind, möchte man manchmal vermeiden. Bekanntlich gibt es dafür beim TRS-80 die Funktion INKEY$, die im BASIC-Programm wie eine String-Variable gehandhabt werden muß. Sie übernimmt ein Zeichen direkt von der Tastatur, aber eben nur eins. Wenn man längere Zahlen auf diese Weise eingeben will, muß man folgendermaßen vorgehen:

```
90  W$=" "
100 FOR I=1 TO N
110 A$=INKEY$: IF A$=" " THEN 110
120 PRINT A$;: W$=W$+A$
130 NEXT: W=VAL(W$)
```

Als Ergebnis erhält man, wenn man beim Eintippen keinen Fehler gemacht hat, eine Zahl mit N Stellen (einschließlich Komma) als Variable W. Benötigt man eine Zeichenkette, kann das W=VAL(W$) natürlich entfallen. Aber wie gesagt, Tippfehler (oder prellende Tasten!) werden nicht verziehen – man bekommt unweigerlich etwas Falsches in den Computer, weil die anderen Schreibfunktionen –ENTER, Rücktaste usw. – nicht funktionieren und man daher nichts korrigieren kann.

Folgendes Programm ahmt nun einige wichtige Funktionen nach, die beim INPUT zusätzlich gegeben sind. Nicht realisiert sind Sprung um 16 Stellen nach vorne (→), Übergang zu Breitschrift (SHIFT →) und Bildschirm Löschen (Clear-Taste). Auch kann man nur eine Zahl und nicht mehrere durch Komma getrennt eingeben. Dies zu bewerkstelligen wäre übrigens für den Leser einmal eine reizvolle Aufgabe! Doch nun das Programm. Es ist als Subroutine ausgeführt, die durch GOSUB 20000 erreicht werden kann und wieder die Variable W liefert.

```
20000   W$=" ": PRINT CHR$ (14);
20010   A$=INKEY$: IF A$=" " THEN 20010
20020   IF ASC(A$)=13 W=VAL(W$): PRINT CHR$ (15):RETURN
```

```
20030   IF ASC(A$)=24 THEN IF LEN(W$)=0 THEN 20010 ELSE
        FOR I=1 TO LEN(W$): PRINT CHR$(8);: NEXT GOTO
        20000
20040   IF ASC(A$)=8 THEN IF LEN(W$)=0 THEN 20010 ELSE
        PRINT A$ ;: W$=LEFT$(W$,LEN(W$)—1): GOTO 20010
20050   PRINT A$;: W$=W$+A$ GOTO 20010
```

Das PRINT CHR$(14) läßt am Anfang der Eingabeprozedur den Cursor erscheinen. Hier kann man eventuell auch die Anforderung zur Dateneingabe einfügen, die auch mit dem INPUT-Befehl auf den Bildschirm gebracht würde. Zeile 20020 gibt dem ENTER seine Funktion zurück, Zeilen 20030 und 20040 verhelfen SHIFT ← und ← zu ihrer normalen Funktion.

Als Resultat kann man (im Rahmen der Fähigkeiten des Computers) beliebig lange Zahlen eingeben, während der Computer zwischendurch etwas anderes macht. Diese anderen Tätigkeiten müßten in Zeile 20010 eingefügt werden, am besten mit GOTO- oder GOSUB-Anweisungen.

Ein Fehler im Level II BASIC

Bei einem so komplexen Maschinenprogramm, wie es das BASIC Level II des TRS-80 darstellt (immerhin nicht weniger als 12K ROM-Bereich!), bleibt auch bei noch so sorgfältiger Entwicklung die Gefahr, daß irgendwelche Fehler irgendwo versteckt zurückbleiben. Bei anderen Microcomputersystemen ist das so und es liegt nahe, auch bei diesem Fabrikat ähnliches zu vermuten. Und tatsächlich, es gibt auch hier einen solchen Fehler, der jedoch nicht von Radio Shack zu erfahren war, sondern vom Autor selbst entdeckt wurde. Dazu möge man folgendes kleines Programm eingeben:

```
10  X=4
20  IF X=4 D=2
30  PRINT D
```

Startet man das Programm, sollte der Computer eigentlich "2" auf dem Bildschirm erscheinen lassen. Statt dessen erscheint "0". Die bedingte Anweisung D=2 wurde also nicht ausgeführt, obwohl die Bedingung X=4 infolge von Zeile 10 erfüllt ist. Wie weitere Tests ergeben, tritt dieser Fehler nur mit den in der obigen Weise verwendeten Variablen D und E auf (sowie mit Variablen, die mit D und E beginnen). Mit A, B, C, F, G usw. arbeitet der Computer korrekt. Auch verschwindet der Fehler, wenn man zwischen X=4 und D=2 ein (an dieser Stelle eigentlich überflüssiges) THEN einfügt.

Aus dem Fehler ergeben sich natürlich eine Menge Folgen — je nach den Einzelheiten des jeweiligen Programmes. Eine Konsequenz ist zum Beispiel, daß in derselben Zeile hinter der fälschlicherweise nicht ausgeführten bedingten Anweisung stehende Befehle ebenfalls unbeachtet bleiben bzw. Anweisungen hinter einem ELSE ausgeführt werden, wenn dies nicht sein soll. Jedenfalls sollte man mit den Variablen D und E vorsichtig sein: THEN verwenden oder vielleicht auch ganz auf diese beiden Buchstaben verzichten.

Und noch eine weitere Lehre ergibt sich hieraus: Auch das Radio Shack Level II BASIC ist nicht ganz ohne Fehler, und wenn ein Programm partout nicht laufen will, obwohl man alles nur denkbare probiert hat, könnte unter Umständen ein weiterer unentdeckter Fehler im Spiel sein. Aber man sollte sich schon sehr sicher sein, alles richtig gemacht zu haben: Fußangeln gibt es in dieser Computersprache einige!

Ungenauigkeiten bei der Addition

Eine der Fußangeln, die beim Radio Shack Level II BASIC leicht zu scheinbar unerklärlichen Fehlern führt, liegt in der begrenzten Genauigkeit, mit der Rechenvorgänge mit Dezimalbrüchen ablaufen. Dies hat zwei Gründe: Einmal natürlich die gegebene Anzahl der Stellen (7 bzw. 17) , dann aber auch die Tatsache, daß der Computer intern Binärzahlen verwendet und dazu jeweils Umwandlungen vornehmen muß. Besonders bei oft wiederholten Rechengängen summieren sich die einzelnen kleinen Fehler zu Abweichungen, die erkennbar werden, obwohl die letzte Stelle der Zahlen jeweils bei der Anzeige weggelassen wird. Aber auch bei bedingten Anweisungen werden die Auswirkungen deutlich: Der Computer vergleicht in solchen Fällen infrage kommende Werte nämlich Bit (Binärziffer) für Bit. Und als Ergebnis unterschiedlicher Rechenvorgänge vorliegende Werte können sich durchaus intern unterscheiden und trotzdem auf dem Bildschirm gleich aussehen. Ein Beispiel: Folgendes Programm –

```
10   FOR X=1 TO 5 STEP .1
20   IF X=3.5 PRINT X
30   NEXT
```

sollte eigentlich als Ergebnis die Zahl 3.5 auf dem Bildschirm erscheinen lassen. Doch das ist nicht der Fall: Es erscheint nichts. Woran liegt das? Das läßt sich an diesem Programm verdeutlichen:

```
10   A=3.5
20   FOR X=1 TO 35: B=B+.1: NEXT
30   FOR X=1 TO 350: C=C+.01: NEXT
40   PRINT " A", " B", "C", ,A, B, C
50   PRINT
60   FOR X=0 TO 3
70   PRINT PEEK (VARPTR(A)+X), PEEK (VARPTR(B)+X), PEEK
     (VARPTR(C)+X)
80   NEXT
```

Wenn man das Programm laufen läßt, entsteht folgendes Bild auf dem Monitor:

A	B	C
3.5	3.5	3.5
0	251	244
0	255	255
96	95	95
130	130	130

A, B und C erhalten alle denselben Wert, nämlich 3.5, aber auf jeweils unterschiedliche Weise: Einmal direkt, dann durch 35malige Addition von 0,1 und schließlich, indem 0,01 350 mal zusammengezählt wird. Die internen Darstellungen der drei Werte unterscheiden sich jedoch, wie man erkennt: Durch Zeile 70 wird der Inhalt der vier Speicherstellen auf den Bildschirm gebracht, die jeweils den Wert A, B und C enthalten.

Jetzt ist auch verständlich, warum der Vergleich in dem ersten Programm versagen mußte. Wie kann man es nun zum Laufen bringen? Doppelt genaue Variable helfen in diesem Fall nicht weiter: Zunächst funktionieren Schleifen mit ihnen überhaupt nicht; auf den Versuch reagiert der Computer mit einer Fehlermeldung (TM ERROR, d. h. ungeeigneter Variablentyp). Außerdem wäre die notwendige Bedingung – Übereinstimmung sämtlicher Binärziffern – auch bei ihnen nicht gegeben. Was man dagegen tun kann, ist die Zahl, auf die es bei dem Vergleich ankommt, vorher in die „genaue" umzuwandeln. Das geht im vorliegenden Fall mit folgendem Ausdruck: Y=INT(X*10+.2)/10. Hat der Wert mehr als eine Stelle hinter dem Komma, muß man mit entsprechend höheren Zehnerpotenzen malnehmen und teilen. Wenn X den Wert 3,45 hat zum Beispiel: Y=INT(X*100+.2)/ 100. Andererseits reicht bei ganzen Zahlen Y=INT(X+.2).

Das " . 2" hat dabei jeweils die Aufgabe, den internen Wert von X sicher über den Wert hinaus anzuheben, der durch die INT-Funktion erreicht werden soll. Bekanntlich wertet sie immer auf die nächste ganze Zahl ab, und wenn der ungenaue Wert geringfügig unter dem genauen liegt, kommt ohne das zusätzliche " . 2 " eine zu niedrige Zahl heraus. Wie man sich überzeugen kann, wird der Wert der Variablen B in unserem Beispiel zu 3,4 statt richtig 3,5.

Ein Bild von dem Fehler, der sich durch wiederholte Addition ergibt, vermittelt am einfachsten dieses Programm:

```
10  FOR X=1 TO 50 STEP .1
20  PRINT X,
30  NEXT
```

Da erscheinen durchaus nicht nur die Zahlen 1, 1.1, 1.2 usw. bis 50, sondern reihenweise auch Werte wie 40.9999, 41.0999, 41.1999 ... Glücklicherweise ergibt sich vergleichsweise selten die Notwendigkeit für solche Konstruktionen. Abzählen wird man üblicherweise mit ganzen Zahlen und da gibt es diese Probleme nicht.

Name

Es gibt im Radio Shack BASIC auch ein Befehlswort, das nicht gebraucht wird: NAME (auf Deutsch: "Name"). Tippt man es ohne weitere Vorkehrungen ein, erscheint die Fehlermeldung, die darauf hinweist, daß es sich um ein Wort des zum Floppy-Disc-System gehörigen, sogenannten Disc-BASIC handelt, nämlich L3 ERROR. Doch auch das stimmt nicht! In Wahrheit ist Name völlig ungebraucht, hat bisher jedenfalls in keinem Radio Shack BASIC eine Funktion. Dies eröffnet nun die Möglichkeit, eine Funktion für NAME selbst zu bestimmen. Doch wie kann das geschehen?

Dazu muß man zunächst wissen, wie die Befehlsausführung im TRS-80 organisiert ist. Das funktioniert jeweils so: Jeder BASIC-Befehl veranlaßt die CPU, sich eine bestimmte, jeweils andere Adresse im Speicher anzuschauen, nämlich die sogenannte Driver-Adresse des Befehls. An diesen Adressen sind wiederum die Anfangsadressen der zu den Befehlen gehörigen Routinen gespeichert, die sich im Festwertspeicher (ROM) des Level II BASICs befinden. Die Driver-Adressen sind dagegen im RAM-Bereich, lassen sich also beliebig mit neuem Inhalt füllen!

Theoretisch könnte man jedem BASIC-Befehl eine andere Bedeutung geben, indem man in seine Driver-Adresse einen anderen Wert lädt. Für die Anweisung USR(x) ist dies ja zum Beispiel auch speziell vorgesehen: In seine Driver-Adresse 16526, 16527 muß man die Anfangsadresse einer Maschinenprogrammsubroutine laden, die aufgerufen werden soll – im Prinzip nichts anderes als die Durchführung eines BASIC-Befehls, dessen Inhalt man aber selbst bestimmt. Dasselbe ist nun auch mit NAME möglich. Seine Driver-Adresse lautet: 16783, 16784 (dezimal). In die erste Position muß wie gewohnt mit POKE das niederwertige Byte der Subroutinen-Anfangsadresse geladen werden, in die zweite das höherwertige Byte. Wie bei den zum Disc-BASIC gehörigen Befehlen enthält die Adresse in Level II normalerweise 301 dezimal, d.h., die Bytes 1 und 45 (dezimal). Dies ist die Anfangsadresse der L3 ERROR-Routine. Diese wird also aufgerufen, wenn man den Speicherinhalt nicht verändert. Der Nachteil von NAME gegenüber USR(x) besteht darin, daß keine Variable im selben Arbeitsgang auf die Subroutine übertragen werden kann. Andererseits läßt sich NAME unverändert auch in DISC-BASIC verwenden. USR(x)-Anweisungen müßten verändert werden, wenn man sich das Floppy-Disc-System dazu kauft und die entsprechenden Programme weiterverwenden will, denn dann gibt es den Befehl in dieser Form nicht mehr.

Abkürzungen in Radio Shack Level II BASIC
Im Gegensatz zum Level I-BASIC (und auch im Unterschied zu anderen BASIC-Versionen) gibt es beim Level II nur zwei Abkürzungen: Für PRINT kann man ein Fragezeichen schreiben, für REM einen Apostroph ('). PRINT und ? sind dabei im computerinternen Befehlscode gleich repräsentiert, d. h. wenn man beim Programmschreiben das Fragezeichen benutzt, erscheint dafür nach dem LIST-Befehl in der betreffenden Programmzeile das Befehlswort PRINT. Das ist vor allem wichtig, wenn man eine Zeile geschrieben hat und sie unmittelbar danach in der EDIT-Betriebsweise des Computers verändern will. Weil der Ersatz des ? durch das PRINT eine Stellenverschiebung innerhalb der Zeile zur Folge hat, die schon bei der Übernahme der Zeile in den EDIT-Betrieb wirksam wird, kann man leicht durcheinandergeraten — die Zeile ist ja in der veränderten Form zunächst nicht zu sehen. Unbedingt empfehlenswert ist daher in diesem Fall der nochmalige Abdruck der Zeile auf dem Bildschirm, ehe man mit dem Verändern beginnt. Dies kann durch LIST (Zeilenzahl) oder, wenn man schon im EDIT-Betrieb ist, durch den den EDIT-Befehl L geschehen.

Im Unterschied zu PRINT und ? bleiben REM und ' als Bezeichnung für Texte, die der Computer nicht beachten soll, auch intern getrennt. REM wird als REM gelistet, ' als '. Wahrscheinlich hat der Hersteller REM nur deshalb in die BASIC-Sprache aufgenommen, weil es gewissermaßen ein Standard-BASIC-Wort ist, daß sich in praktisch allen anderen Versionen wiederfindet (genau wie das LET für die Wertzuweisung, das ja beim TRS-80 überflüssig, aber trotzdem vorhanden ist). In der Praxis wird man wohl nur das kürzere ' verwenden.

An dieser Stelle ist es vielleicht ganz passend, einige Bemerkungen über das ' als Programmierhilfe zu machen. Abgesehen von seinem eigentlichen Zweck, nämlich erklärende Bemerkungen in einem Programm unterzubringen, die der Computer nicht beachten soll, läßt sich der Apostroph auch noch beim Programmschreiben verwenden und zwar auf zweierlei Weise:

Soll zum ersten eine bestimmte Programmzeile bei einem Testlauf des Programms übersprungen werden, kann man einfach ein ' am Anfang einfügen. Das ist wesentlich einfacher als einen Sprungbefehl an die vorherige Zeile anzufügen bzw. einen vorhandenen zu ändern — oder

gar mehrere an verschiedenen Stellen des Programms, wenn die betreffende Zeile von verschiedenen Stellen aus über Sprunganweisungen aufgerufen wird. Auch läßt sich nur mit dem ' ein Teil einer Zeile (nämlich der darauffolgende) auf einfache Weise ausschalten, ohne daß er dazu gelöscht werden muß.

Die andere Anwendung besteht darin, durch den Apostroph als einzigen Inhalt einer Zeile die Zeilennummer im Programm zu halten, auch wenn der ursprüngliche Zeileninhalt nicht mehr gebraucht und darum gelöscht wurde. Dies kann von großem Wert sein, wenn die Zeile mit einem Sprungbefehl angesprochen wird. Dieser Sprung funktioniert dann nämlich immer noch, wenn die betreffende Zeile eigentlich schon aus dem Programm entfernt wurde. Man vermeidet damit, daß das Programm solange nicht läuft, bis man auch die Sprungadresse in geeigneter Weise geändert hat.

Platzsparendes Programmieren von Graphik-Zeichenketten

Platzsparendes Programmieren von Graphik-Zeichenketten
Daß mit Zeichenketten (Strings) elegantes Programmieren komplizierter Bildschirmgraphik-Darstellungen möglich ist, wenn man graphische Code-Zahlen (128–191) Zwischenraum-Codezahlen (192–255), Steuer-Codezahlen (8–31) in CHR$- und STRING$-Funktionen anwendet, ist bereits dargestellt worden. Da dies verhältnismäßig viel Speicherraum in Anspruch nimmt– erforderlich ist neben dem Programmtext selbst auch noch Platz im String-Bereich des RAM-Speichers –, kann man selbst bei 16-K-Maschinen mit längeren Programmen in Platznot kommen. Doch gibt es noch eine andere Methode, die entsprechenden Zeichenketten zusammenzustellen, so daß sie im Programmtext selbst stehen und dort nur ihrer tatsächlichen Länge gemäß Platz beanspruchen. Allerdings kostet das Programmieren dabei ein wenig zusätzliche Mühe.

Das Problem besteht darin, daß man Steuer- und Graphik-Codeszahlen nicht von der Tastatur aus erreichen kann, so daß es nicht ohne weiteres möglich ist, entsprechende Strings wie mit Buchstaben oder Zahlen einfach in der Form A$="OTTO MUELLER" zu definieren. Um denselben Effekt zu erzielen, kann man jedoch folgendermaßen vorgehen:

Zunächst legt man innerhalb des vorgesehenen Programms die für Graphik-Darstellungen bestimmten Zeichenketten in ihrer geplanten Länge fest, indem man sie mit beliebigen Buchstaben oder einfach mit

Leerstellen füllt, zum Beispiel A$=" ....", B$="..........." usw. Wenn das Programm in dieser Weise fertiggestellt wurde, schreibt man zusätzlich folgendes Hilfsprogramm:

```
50000  K=PEEK(VARPTR(A$)+1)+256*PEEK(VARPTR(A$)+2)
50010  FOR I=0 TO 10: READ C: POKE K+I,C: NEXT
50020  DATA ......
```

Damit kann man dann nacheinander die einzelnen Strings mit den entsprechenden Codezahlen füllen — sie werden durch die POKE-Anweisung direkt in den Speicherplatz des Strings gebracht, der ja durch die vorherige Definition der Strings im eigentlichen Programmtext offengehalten wurde. Für jede Zeichenkette muß dabei der entsprechende Variablen-Name in die beiden VARPTR-Funktionen in Zeile 50000 eingefügt werden, und die Lese- und POKE-Schleife in Zeile 50010 muß an die Länge des jeweiligen Strings angepaßt werden (Endzahl um 1 kleiner als die Anzahl der Stellen, da die Schleife mit 0 beginnt). In der DATA-Zeile schließlich müssen die gewünschten Code-Zahlen stehen. Am Ende muß nur noch das Hilfsprogramm wieder gelöscht werden und man hat ein Programm mit äußerst rationell gespeicherten Graphik-Strings.

Ein etwas merkwürdiges Aussehen nimmt allerdings der Programmtext als Ergebnis dieser Manipulation an: Graphik- und Zwischenraum-Codezahlen sind in der computer-internen Organisation BASIC-Befehls-Codezahlen und so werden die beim Listen dann auch auf den Bildschirm gebracht.

Hat man zum Beispiel die ersten fünf Graphik-Codezahlen verwendet, nämlich 129, 130, 131, 132 und 133, sieht die entsprechende String-Definition nach der Anwendung des Hilfsprogrammes nicht mehr so aus: A$=".....", sondern so: A$="FORRESETSETCLSCMD". A$ besteht jetzt also aus den fünf BASIC-Wörtern FOR,RESET,SET,CLS und CMD (der letzte gehört allerdings zum sogenannten Disc-BASIC, das man mit dem Erweiterungsteil und dem Floppy-Disc-System verwenden kann). Wichtig ist dabei, daß man durch einfaches Schreiben der Buchstabenfolge nicht die Graphik-Zeichenkette erhält, sondern eben nur die Reihe von Buchstaben.

Die Aneinanderreihung von BASIC-Wörtern, die man bei dem Listen des Programmtextes erhält, ist natürlich viel länger als das ursprüngliche String und kann im Grenzfall sogar die größtmögliche Länge einer Programmzeile, nämlich 255 Stellen sprengen, so daß die Zeile überhaupt nicht vollständig auf dem Bildschirm erscheint. Für das Programm ist das jedoch ohne Belang, und man muß sich vor Augen halten, daß intern sehr viel weniger Speicherplatz benötigt wird, als würde man die Zeichenkette in herkömmlicher Weise bilden.

Noch seltsamer wird das Listen des Programms, wenn man Strings mit Kontroll-Codezahlen gefüllt hat. Diese Funktionen werden beim Listen nämlich schlicht ausgeführt, so daß das Bild des Programmtextes völlig durcheinandergeraten kann. Da es keine Worte oder andere Zeichen für diese Codezahlen gibt, werden sie andererseits auch nicht als Inhalt des jeweiligen Strings sichtbar und es erscheint A$=" ", wenn nur Kontroll-Codezahlen verwendet wurden. Natürlich kann das für jemanden, der nicht weiß, wie das Programm zustande gekommen ist, sehr verwirrend sein.

Noch ein wichtiger Hinweis: Programmzeilen, die die mit dem Hilfsprogramm gebildeten Zeichenketten enthalten, dürfen anschließend nicht mehr in der Edit-Betriebsart abgeändert werden. Dabei geht nämlich stets – auch wenn man nur "ENTER" drückt, ohne tatsächlich etwas verändert zu haben – der spezielle Charakter des oder der Strings verloren. Man müßte dann den Bildungsvorgang wiederholen.

„Programmieren in Maschinensprache und Assembler"

Programmieren in Maschinensprache und Assembler
Dieses Kapitel ist eigentlich garkein Kapitel über den TRS-80, sondern über sein Herz, der Zentraleinheit (CPU) Z-80 von Zilog. Denn mit Maschinenprogrammen erreicht man sie direkt, ohne den „Umweg" des BASIC-Interpreters, der im Grunde nichts weiter ist, als ein (Maschinen-)Programm zur Übersetzung der BASIC-Programmsprache in Maschinensprache. Wie schon erwähnt, ersparen Maschinenprogramme diese Übersetzung und werden deshalb wesentlich schneller verarbeitet. Als Preis für diesen Vorteil muß man allerdings erhebliche Mehrarbeit beim Erstellen der Programme in Kauf nehmen. Während BASIC als „höhere" Programmiersprache nämlich problemorientiert ist — was man den Computer tun lassen will, kann man mehr oder weniger direkt formulieren —, richtet sich der Aufbau eines Programms in Maschinensprache nach der Arbeitsweise des jeweiligen Computers: Jede Einzelheit einer Aufgabe muß dem Rechner einzeln vorgegeben werden. Im Prinzip besteht also die Maschinenprogrammierung darin, Maschinenbefehl an Maschinenbefehl zu reihen, die jeweils nur einen Schritt in der Arbeit der CPU bewirken.

Normalerweise kann dies nach zwei verschiedenen Methoden geschehen: Als direkte Maschinenprogrammierung oder in einer sogenannten Assemblersprache. Im ersten Fall verwendet man gewöhnlich als Hilfsmittel einen sogenannten Monitor. Das ist ein Hilfsprogramm, das es erlaubt, sozusagen in das Innenleben des Computers hineinzuschauen: Der Inhalt von CPU-Registern sowie Speicher-Abschnitten wird auf dem Bildschirm abgebildet, man kann Maschinenbefehle eingeben und in bestimmte Speicherbereiche befördern lassen. Die geschriebenen Maschinenprogramme kann man für Testzwecke Schritt für Schritt oder bis zu einem vorgewählten Punkt laufen lassen und noch einiges mehr. Da-

bei werden durchweg Hexadezimalzahlen verwendet, deren Ziffern von 0 bis 9 sowie die ersten 6 Buchstaben des Alphabets umfassen. Der TRS-80 enthält im Gegensatz zu anderen Microcomputern keinen fest installierten Monitor – nur die Möglichkeit, vorhandene Maschinenprogramme vom Cassettengerät zu laden und zu starten. Doch bietet Radio Shack einen Monitor zu günstigen Bedingungen als Maschinenprogramm-Cassette an – das „T-Bug"-Programm. Auch das Disc-Betriebssystem enthält einen Monitor.

Die andere, in vieler Hinsicht komfortablere Möglichkeit besteht darin, ein Assembler-Programm zu verwenden. Die Assembler-Sprache ist (im Gegensatz zum BASIC) ebenfalls jeweils typisch für die CPU eines Computers, und so auch für den Z-80, den der TRS-80 verwendet. Wesentlichstes Merkmal aller Assemblersprachen ist jedoch, daß die Maschinenbefehle als leichter merkbare (allerdings englische) Abkürzungen dargestellt werden. Nach wie vor entspricht ein Programmbefehl einem Ausführungsschritt der CPU; jeder etwas komplexere Vorgang, der im BASIC-Programm vielleicht nur einen Befehl oder eine Funktion erfordert, muß auch beim Assembler-Programmieren aus unter Umständen zahlreichen dieser Schritte zusammengesetzt werden.

Abgesehen davon bieten Assemblersysteme, die das in der Assemblersprache geschriebene Programm in Maschinensprache umwandeln, im allgemeinen eine Reihe nützlicher Eigenschaften.

In Assembler geschriebene Programme haben (wie beim BASIC) Zeilennummern. Die einzelnen Zeilen bestehen immer aus Zeilennummer, Befehlscode und ein oder zwei Operanden, an dem bzw. denen die Anweisungen ausgeführt werden sollen. Daran kann sich bei Bedarf, durch ein Semikolon abgetrennt, eine Erläuterung anschließen, die für das Programm keine Bedeutung hat. Um tatsächliche Speicheradressen bzw. Programmstellen für Verzweigungen, bedingte Anweisungen usw. braucht man sich nicht zu kümmern: Dafür gibt es frei wählbare Markierungswörter (sogenannte Labels), die man vor den Befehlscode in die jeweilige Zeile einfügen kann. Für Teile, die in einem Programm mehrmals vorkommen, lassen Assemblersysteme meist sogenannte Makros zu. Sie werden jeweils nur einmal programmiert und dann bei der Umwandlung des Assemblerprogramms in das Maschinenprogramm an den gewünschten Stellen eingefügt. Damit sind Makros keine Unter-

programme, da sie im Maschinenprogramm tatsächlich jedesmal im Verlauf des Programmes vorhanden sind, wenn sie ausgeführt werden sollen.

Die von Zilog für ihre CPU geschaffene Assemblersprache bzw. das zugehörige Assemblersystem erlaubt unter anderem Verwendung von Binär-, Oktal-, Dezimal- und Hexadezimalzahlen. Möglich sind Programm-Testläufe; es gibt ein Editsystem zur Programmbearbeitung und es werden beim Übersetzen in den Maschinen-Code 17 verschiedene Fehler erkannt und angezeigt. Auch ist die Möglichkeit gegeben, einfache Rechenvorgänge mit ganzen Zahlen direkt zu programmieren.

Für den TRS-80 mit 16K Speicherplatz und Level I oder Level II BASIC bietet Radio Shack eine etwas abgemagerte Version des Z-80-Assemblers als Maschinenprogramm-Cassette an. Makros werden nicht bearbeitet, die direkten Rechenvorgänge sind eingeschränkt, man kann nicht alle Arten von Zahlen verwenden usw. Dafür gibt es eine Bearbeitungsmöglichkeit für Programmtexte, die stark an Betriebssystem und Edit-Betriebsart des Computers angelehnt ist und nur wenig Umlernen erfordert. Hat man seinen TRS-80 bis zu den Floppy-Disc-Geräten erweitert, kann man auch das vollständige Assembler-System von Zilog einsetzen, das es auf einer Diskette zu kaufen gibt.

Es sollte an dieser Stelle vielleicht noch einmal erwähnt werden, daß die Z-80-CPU als Weiterentwicklung der bekannten 8080-CPU bis auf unbedeutende Kleinigkeiten vollständig „aufwärtskompatibel" mit ihr sowie mit ihrer späteren Version 8085 ist. Das heißt, alle für Microcomputer-Systeme mit 8080-CPU geschriebenen Maschinenprogramme laufen auch auf dem TRS-80, vorausgesetzt, die übrigen Voraussetzungen wie Speicherplatz-Belegung, Ein- und Ausgabewege usw. stimmen ebenfalls überein.

In der Praxis beweisen Maschinenprogramme in Verbindung mit BASIC-Programmen ihren Wert, wenn es darum geht, eine Aufgabe auszuführen, die für das BASIC zu langsam oder in anderer Hinsicht schlecht geeignet ist. Dazu dient die Möglichkeit, ein Maschinenprogramm als Subroutine vom BASIC-Programm aus mit der Funktion USR aufzurufen. Dafür gibt es in diesem Buch auch ein Beispiel, nämlich die Erzeugung von Geräuscheffekten und ganzen Melodien mit dem TRS-80. Die ein-

fachste, aber umständlichste Prozedur, ein BASIC-Programm mit einer zugehörigen Subroutine in den Computer zu bringen, besteht darin, zuerst das Maschinenprogramm von der Cassette in einen reservierten Speicherbereich zu laden und dann das BASIC-Programm zu laden.

Rationeller ist es jedoch, das Laden des Maschinenprogramms vom BASIC-Programm selbst vornehmen zu lassen. Wenn man das Maschinenprogramm in irgend einer Weise in den Computer hineinbekommen hat – entweder selbst geschrieben oder von der Cassette geladen – kann man sich den belegten Speicherbereich mit entsprechenden PEEK-Anweisungen anschauen und die Reihe von Zahlen die das Programm bildet, als DATA-Zeilen in das BASIC-Programm aufnehmen. POKE-Befehle können es dann nach dem Start des BASIC-Programms in seine Position bringen. Der entsprechende Speicherbereich muß dazu natürlich beim Anschalten des Computers reserviert worden sein.

Nimmt man gewisse Nachteile beim Computerbetrieb in Kauf, kann man Maschinenprogramm-Subroutinen sogar in Bereichen des BASIC-Programmtextes unterbringen, die mit Zeichenketten entsprechender Länge freigehalten werden. Dafür gibt es in diesem Buch ebenfalls ein Beispiel. Dieser Trick erübrigt sogar das Reservieren eines gesonderten Speicherbereichs für das Maschinenprogramm.

In diesem Zusammenhang ist Folgendes sehr wichtig: Es gibt grundsätzlich zwei Arten von Maschinenprogrammen: Die einen enthalten Sprunganweisungen zu fest vorgegebenen Adressen innerhalb des Programms. Sie heißen „nicht relokierbar", denn es ist offensichtlich nicht gleichgültig, an welcher Stelle im Speicher sie sich befinden. Die andere Art enthält innerhalb des Programmbereichs nur relative Sprünge, d. h. Sprünge, um eine bestimmte Anzahl von Adressen vorwärts oder rückwärts vom jeweiligen Inhalt des Befehlszählers, der Speicheradressen beim Programmablauf zählt, entsprechend den Zeilenzahlen bei BASIC-Programmen. Solche Programme sind relokierbar, bei ihnen ist es gleichgültig, wo sie im Computerspeicher untergebracht sind. Man sollte meinen, daß wegen dieses offensichtlichen Vorteils nur die zweite Programmart Bedeutung hat. Doch leider sind die Möglichkeiten für relative Sprünge begrenzt. Sie können nur bis zu 126 Speicherstellen weit rückwärts und 129 Speicherstellen nach vorne ausgeführt werden.

Auch hat die Unterscheidung nicht sehr große Bedeutung, wenn man ein Maschinenprogramm von einer Cassette lädt. Dabei wird die Anfangsadresse immer mitgeliefert, liegt also fest. In diesem Fall und auch, wenn man eine Subroutine mit Hilfe eines BASIC-Programmes einrichtet, muß die Anfangsadresse bekannt sein, damit sie in den Platz für die Sprungadresse der USR-Funktion geladen werden kann. Wenn sie infolge der Nicht-Relokierbarkeit des Maschinenprogramms nicht frei gewählt werden kann, ist dies keine allzugroße Erschwernis. Allerdings muß man sich bei der Bemessung des reservierten Speicherbereichs nach der (festen) Anfangsadresse richten und kann, falls sie nicht ganz ideal gewählt wurde, nicht den kleinsten, gerade noch erforderlichen Speicherplatz freihalten.
Das Maschinenprogramm innerhalb des BASIC-Programmtextes unterzubringen wird natürlich ziemlich schwierig, wenn es an eine feste Anfangsadresse gebunden ist. Ehe das BASIC-Programm fertig ist, weiß man ja nicht, wo die Zeichenkette im Speicher dann später abgelegt wird, welche das Maschinenprogramm aufnehmen soll. Man müßte nach Fertigstellung des BASIC-Textes das Maschinenprogramm so abändern, daß alle Sprunganweisungen an ihr vorgesehenes Ziel führen, und dürfte das BASIC-Programm anschließend (zumindest im Bereich vor der Zeichenkette) keinesfalls mehr verändern.

Einen interessanten Trick, wie man ein eigentlich nicht relokierbares Maschinenprogramm dennoch für jeden beliebigen Speicherort geeignet machen kann, zeigt übrigens das Unterprogramm zum Laden der Tonerzeugungs-Subroutine in den Geräuscheffekt- und Musikprogrammen dieses Buches: Wenn man sich die DATA-Zeilen ansieht, bemerkt man eine Reihe negativer Zahlen. Das sind nun tatsächlich relative Sprungweiten, allerdings stets auf die Anfangsadresse der Subroutine bezogen. Die Programmschleife zum Übertragen der einzelnen Zahlen in den vorgesehenen Speicherbereich erkennt dies an dem Minuszeichen (Zeile 50080), rechnet die Relativwerte mit Hilfe der Anfangsadresse in Absolutwerte um (Zeile 50120) und bringt sie nach der Zerlegung in zwei Bytes (Zeile 50130) an ihren Platz (Zeile 50140). So befindet sich schließlich ein Maschinenprogramm mit lauter absoluten Sprunganweisungen an irgendeiner beliebigen Stelle im Speicher und funktioniert trotzdem. Bei der Programmversion mit der Subroutine in einem geschützten Speicherbereich kann dieser beliebig, zum Beispiel auch so klein wie nur irgend möglich, gewählt werden – die Position der Subroutine orientiert sich stets an dem Beginn des geschützten Bereichs, mit entsprechend angepaßten Sprungbefehlen.

```
        ORG     7FC4H           ;NEAR END OF 16K MEM.
START   LD      HL,SOUND        ;SETUP USR(Ø) TO
        LD      (16526),HL      ;POINT TO SOUND
        JP      1A19H           ;JUMP TO BASIC
SOUND   LD      HL,(SBUFF)      ;GET STRING DOPE VCTR
        LD      A,(HL)          ;GET LEN
        OR      A               ;IS IT ZERO?
        PUSH    AF              ;SAVE LEN AND TST RSLT
        INC     HL              ;NEXT ITEM
        LD      E,(HL)          ;LSB OF STR ADDR.
        INC     HL              ;NEXT ITEM
        LD      D,(HL)          ;MSB OF STR ADDR.
        EX      DE,HL           ;STR ADDR IN HL
NEXT    POP     AF              ;NO. CHRS REMAINING
        RET     Z               ;NO MORE
        DEC     A               ;MUST BE 2 OR MORE
        RET     Z               ;IT'S NOT
        DEC     A               ;COUNT 2ND IN PAIR
        PUSH    AF              ;SAVE REMAINING CNT.
        LD      D,(HL)          ;GET DURATION CHR.
        LD      E,Ø             ;(*256)
        INC     HL              ;POINT TO PITCH CHR.
        LD      C,(HL)          ;GET IT
        INC     HL              ;NEXT CHR, IF ANY
TONE    LD      A,1             ;SQUARE WAVE PLUS
        CALL    HOLD            ;SUSTAIN IT
        JR      Z,NEXT          ;ALL DONE
        LD      A,3             ;SQUARE WAVE MINUS
        CALL    HOLD            ;SUSTAIN IT
        JR      NZ,TONE         ;MORE OSCILLATIONS
        JR      NEXT            ;ALL DONE
HOLD    OUT     (ØFFH),A        ;TO CASSETTE PORT
        LD      B,C             ;PITCH PERIOD
LOOPB   DEC     DE              ;DECREMENT DURATION
        LD      A,D             ;TEST FOR ZERO
        OR      E               ;     "
        RET     Z               ;DONE IF SO
        DJNZ    LOOPB           ;CNT DOWN PERIOD & LOOP
        RET                     ;DONE COUNTING
SBUFF   DEFS    2               ;THIS IS LOC. 32766
        END     START           ;FOR AUTOSTART
```

Figure 2: The tone-generating program SOUND.

Beispiel Assemblerprogramm (ohne Zeilennummern)
Dieses Programm erzeugt Töne auf dem TRS-80. Ausgang ist der AUX-Stecker vom Cassetteninterface.

Ein kleiner Versuch soll den Geschwindigkeitsunterschied zwischen BASIC- und Maschinenprogrammen zeigen. Dazu wollen wir eine einfache „Tätigkeit" für den Computer wählen, das Weißmalen des Bildschirms in möglichst kurzer Zeit.

Die mehr oder weniger schnellste Methode, dies mit BASIC zu bewerkstelligen, dürfte dieses Programm darstellen:

```
10  CLEAR 255: PRINT CHR$(23) STRING$ (255,191)STRING$
    (255,191) CHR$ (191) ; : POKE 16382, 191
20  GOTO 20
```

Nachdem man dies ausprobiert hat und sich ein Bild von der Schnelligkeit gemacht hat, mit der der Bildschirm weiß färbt, kann man mit dem Maschinenprogramm fortfahren. Es wird, um die Sache möglichst einfach zu machen, ebenfalls mit einem kleinen BASIC-Programm in den Computerspeicher geschrieben. Zuerst ist allerdings eine kleine Vorbereitung nötig: Man muß den TRS-80 (der 16K Speicherkapazität haben muß) kurz aus- und dann wieder einschalten (oder SYSTEM gefolgt von / und ENTER eingeben). Auf die Frage MEMORY SIZE? ist die Zahl 20479 einzugeben. Dann kann man folgendes Programm einschreiben oder, falls bereits vorhanden, vom Cassettengerät einlesen:

```
10  FOR X=20480 TO 20507
20  READ A: POKE X,A
30  NEXT
40  DATA  33,0,60,17,1,60,  1,0,4,54,191,237,176,6,5,33,255,255,
    43,124,181,194,18,80,16,245,195,0
```

Läßt man dieses Programm laufen, wird das aus den Zahlen in der DATA-Zeile bestehende Maschinenprogramm in die Speicherpositionen 20480 bis 20507 übertragen. Gestartet wird es anschließend durch SYSTEM und /20480. Als Ergebnis wird der Bildschirm schlagartig weiß.

Er bleibt einige Sekunden so, dann ist das Programm abgearbeitet, und der Computer fällt in die Anschaltroutine zurück. Dadurch wird zwar das BASIC-Programm gelöscht, nicht aber das Maschinenprogramm, so

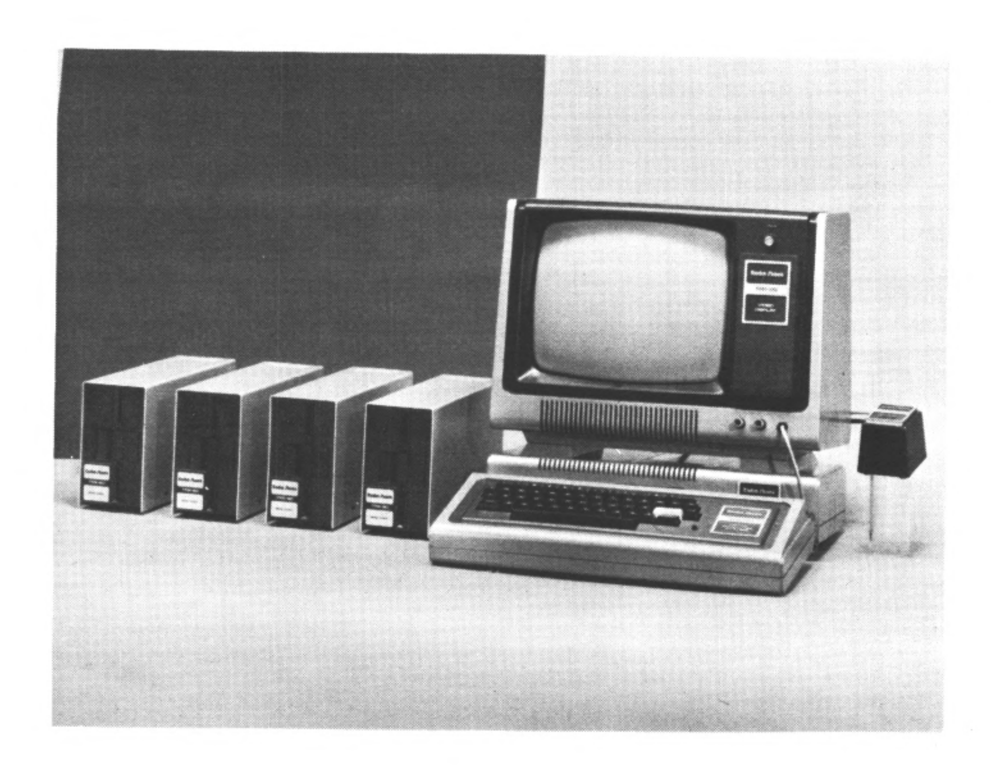
daß man es noch beliebig oft starten kann. Da kein BASIC-Programm mehr in den Speicher soll, braucht man den Bereich des Maschinenprogramms auch nicht mehr zu schützen und kann die Einschaltroutine jetzt durch einfaches Drücken von ENTER beenden.

Die in der Speicherposition 20490 stehende Zahl 191 stellt übrigens genau wie in entsprechenden BASIC-Programmen ein Graphik-Zeichen dar, nämlich das weiße Feld, das 1024 mal geschrieben den weißen Bildschirm ergibt. Durch Ändern dieser Zahl mit einem entsprechenden POKE-Befehl kann man das Maschinenprogramm den Bildschirm auch mit beliebigen Zahlen, Buchstaben usw. füllen lassen.

Das TRS-80 Floppy Disc-System

Es war bereits davon die Rede, daß das Erweiterungsbauteil „expansion interface" des TRS-80 dem Computer eine neue Dimension von Anwendungen in Richtung auf ein schon echt kommerzielles System erschließt. Zum einen läßt es sich mit bis zu 32K weiterem freien Speicherraum ausstatten. Dann ermöglicht es den Anschluß eines zweiten Cassettenrecorders. Doch vor allem enthält das Gerät die Steuerschaltung für das Floppy Disc-System des Computers. Und dieses kann – von einigen weiteren Einzelheiten abgesehen – als die eigentliche Funktion des Apparates angesehen werden, nachdem bereits der Erweiterungsanschluß der Computer-Grundeinheit sich als Ausgang für Drucker und andere Zusatzgeräte eignet.

An das Expansion Interface lassen sich bis zu vier Laufwerke anschließen, die als Datenspeicher-Vorrichtungen dünne, biegsame, runde Scheiben aus magnetisierbarem Material (sog. Disketten) mit ca. 13 cm

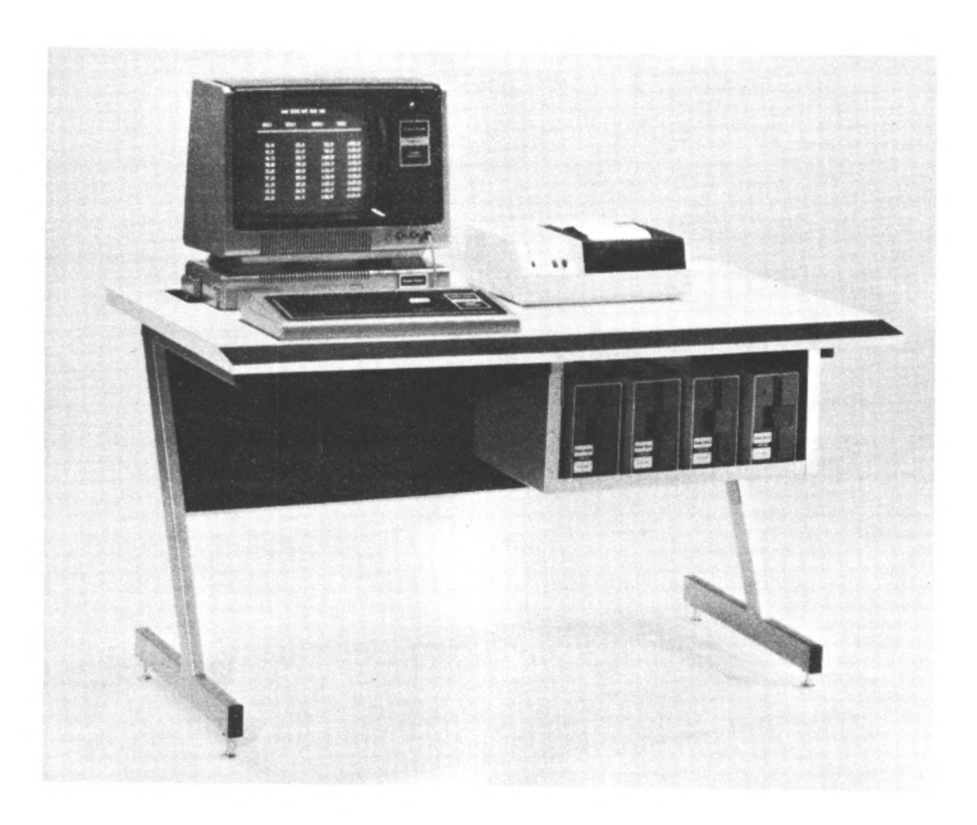


TRS-80 mit Expansion-Interface und 4 Floppy-Disk-Laufwerken

Durchmesser aufnehmen – sogenannte Mini-Floppy Discs. Die Scheiben können 89 K Byte Informationen aufnehmen, und die ist ganz wesentlich schneller erreichbar, als auf den Tonbandcassetten. Eine der Scheiben – wenn man nur ein Laufwerk hat, die einzige verwendete – enthält auch das TRS-80-Disc BASIC sowie das Floppy Disc-Betriebssystem, ein Maschinenprogramm (Name: TRSDOS). Das Disc BASIC bietet, außer den besonderen Anweisungen für den Floppy-Disc-Betrieb, einige zusätzliche Möglichkeiten, die der Computer ohne Erweiterungsteil nicht hat, das TRSDOS, unter anderem eine Uhr, die rechts oben auf dem Bildschirm eingeblendet werden kann. Die zugehörigen Schaltungen sind im Expansion Interface eingebaut und inzwischen gibt es auch ein kleines Maschinenprogramm, mit dessen Hilfe man die Uhr auch ohne Floppy Disc-System erscheinen lassen kann. Zum Gebrauch werden Disc BASIC und auch das TRSDOS stets von der Magnetscheibe in den Speicher des Computers geladen und nehmen dort knapp 10K Raum ein. Wenn man die Floppy Discs verwenden will, empfiehlt es sich daher, den Erweiterungsteil zumindest mit 16K Speicherraum auszustatten. Sonst bleiben (von den knapp 16K des Computer-Hauptteils) nur noch vergleichsweise magere 6K für Programme übrig.

Selbstverständlich kann man Daten geordnet oder ungeordnet auf die Mini-Floppies schreiben und wieder lesen. Dazu werden sogenannte Files auf den Scheiben verwendet, gewissermaßen Ablageplätze für gewisse Datenmengen. Das Betriebssystem TRSDOS tritt mehr oder weniger an die Stelle des normalen Computer-Betriebssystems, was das Floppy Disc-System angeht. Damit kann man zum Beispiel BASIC- oder Maschinenprogramme auf Disketten speichern und wieder laden, Scheiben kopieren usw. Eine interessante Möglichkeit besteht darin, einzelne Files oder auch ganze Disketten mit einem Codewort zu schützen. Je nach Umfang des Schutzes lassen sich dann nur bestimmte Funktionen ausführen, ohne daß man dazu das Codewort angeben muß.

Mit dem voll ausgebauten Floppy Disc-System verfügt der TRS-80 über mehr als 350 K verhältnismäßig schnell zugänglicher Speicherkapazität. An große Computersysteme – z. B. Minicomputer – reicht dies natürlich noch nicht heran. Ernsthaften kommerziellen Einsatz erlaubt es jedoch allemal. Darüber hinaus bietet sich jetzt auch die Gelegenheit,



Komplettes Computersystem für den Geschäftsbereich

die gewissermaßen fest in den Computer eingebauten Wege zu verlassen und beispielsweise andere Programmiersprachen zu verwenden, als das vorhandene BASIC. Tatsächlich ist unter anderem ein Maschinenprogramm auf Disketten erhältlich, mit dessen Hilfe man eine FORTRAN-Version auf dem TRS-80 einsetzen kann.

Allerdings dürfte dies alles doch wohl ein wenig über den Rahmen hinausgehen, den man vernünftigerweise für den privaten Einsatz eines Microcomputers als gegeben voraussetzen kann.

Der Erweiterungs-Ausgang des TRS-80 (Expansion Port)

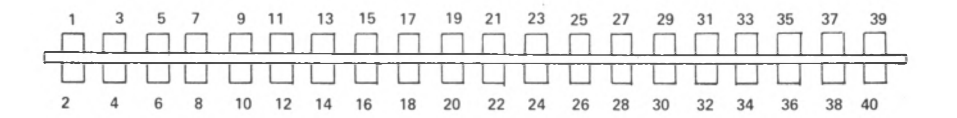
Rechts (von hinten gesehen) an der Rückseite des Computergehäuses befindet sich unter einer abnehmbaren Klappe der Erweiterungs-Busanschluß des TRS-80. Er ist als 40-poliger Platinenanschluß ausgeführt und läßt sich mit einem entsprechenden Platinenstecker (AMP P/N 88103-1 oder ähnliches) abgreifen. Vom Hersteller ist der Busanschluß für den Erweiterungsbauteil gedacht. Doch kann der TRS-80-Freund auch seine eigenen Erweiterungen (Steuerungen, Zusatzgeräte) hier anschließen. Dazu sollen an anderer Stelle noch einige Erklärungen folgen. Hier zunächst die Lage und Bezeichnungen der einzelnen Anschlüsse:

Der Bus-Ausgang des TRS-80

Anschlußnummer	Signalname	Beschreibung
1	RAS'	Taktausgang Adressenzeile
2	SYSRES'	RESET-Ausgang
3	CAS'	Taktausgang Adressenspalte
4	410	Adressenausgang
5	A12	Adressenausgang
6	A13	Adressenausgang
7	A15	Adressenausgang
8	GND	Erde
9	A11	Adressenausgang
10	A14	Adressenausgang
11	A8	Adressenausgang
12	OUT'	Peripherie-Schreib-Taktsignal
13	WR'	Speicher-Schreib-Taktsignal
14	INTAK'	Interrupt-Bestätigung
15	RD'	Speicher-Lese-Taktsignal
16	MUX	Kontrollsignal Adressmultiplexer
17	A9	Adressenausgang
18	D4	Datenleitung
19	IN'	Peripherie-Lese-Taktsignal
20	D7	Datenleitung
21	INT'	Interrupt-Eingang (maskierbar)
22	D1	Datenleitung
23	TEST'	Testeingang

24	D6	Datenleitung
25	A0	Adressenausgang
26	D3	Datenleitung
27	A1	Adressenausgang
28	D5	Datenleitung
29	GND	Erde
30	D0	Datenleitung
31	A4	Adressenausgang
32	D2	Datenleitung
33	WAIT'	Warte-Eingang
34	A3	Adressenausgang
35	A5	Adressenausgang
36	A7	Adressenausgang
37	GND	Erde
38	A6	Adressenausgang
39	+5V	5-V-Anschluß, bei Level II-Geräten auf Erdpotential
40	A2	Adressenausgang

Anmerkung: Die mit Apostroph versehenen Signale haben als logisches „Wahr" niedriges Potential.



Nummerierung der Bus-Anschlußleitungen (von der Rückseite des Computergehäuses aus gesehen).

Die Erklärungen zu den einzelnen Anschlüssen:

Adressenausgänge

Von diesen Ausgängen gibt es insgesamt 16: A0 bis einschließlich A15. Auf A0 befindet sich das geringstwertige Adressenbit, auf A15 das höchstwertige. Diese Leitungen benutzt (im Normalfall) nur die zentrale Recheneinheit (CPU), die damit Speicherstellen anspricht, um Information zu lesen oder zu schreiben. Insgesamt sind 65 536 (2^{16} oder 16K) verschiedene Adressen möglich. WICHTIG! Jede Adressenleitung darf höchstens mit **einem** Logikanschluß (Transistor-Transistor-Logik-TTL) belastet werden!

Datenausgänge
Es gibt 8 Datenausgänge, die mit D0 bis D7 bezeichnet sind. D0 trägt das geringstwertige, D7 das höchstwertige Datenbit. Die Datenausgänge sind bidirektional, d. h. auf ihnen bewegen sich Informationen von der Zentraleinheit (CPU) zu Speicher, Ausgängen, Anschlußgeräten usw. sowie auch umgekehrt. Das bedeutet für Anschlußgeräte, daß diese stets für jede Datenrichtung mit einem Tri-State-Buffer ausgerüstet sein sollten, also mit einem Datenpuffer, dessen Anschlüsse jeweils niedrige und hohe Spannung (logisch "0" und "1") sowie zusätzlich einen hochohmigen, „abgeschalteten" Zustand einnehmen können.

Taktausgang für Adressenzeile
Diese Steuerleitung (Bezeichnung: RAS') ist ebenso wie der Taktausgang für die Adressenspalte (siehe unten) erforderlich, weil die CPU eigentlich nur Adressen mit 8 Bit Länge ausgeben kann und die 16 Bit lange Adressen dieses Computersystems deshalb gewissermaßen in zwei 8-Bit-Portionen unterteilt. RAS' liegt normalerweise auf logisch"1"(hohe Spannung) und wird von der CPU während der ersten Hälfte einer Adressenausgabe auf logisch "0" geschaltet. Für die dynamischen RAMs des TRS-80 dient dieses Signal gleichzeitig zur (bei dieser RAM-Bauart notwendigen) Auffrischung des Speicherinhalts.

Taktausgang für Adressenspalte
Dieser Ausgang (CAS') wird während der zweiten Hälfte einer Adressenausgabe durch die Zentraleinheit auf logisch "0" geschaltet, so daß die Speichereinheiten diesen Adressenteil vom ersten unterscheiden können (siehe oben).

Kontrollsignal für den Adressmultiplexer
Dieses Signal (MUX) schaltet den Multiplexer um, der die erste und zweite Adressenhälften zu je 8 Bit getrennt zu den Speichereinheiten führt. Dieses Signal arbeitet mit RAS' und CAS' zusammen (s. oben).

RESET-Ausgang
Das Signal auf dieser Leitung (SYSRES'-System RESET) liegt nur dann auf niedriger Spannung, wenn der Computer angeschaltet oder der RESET-Knopf gedrückt wird. SYSRES' kann man verwenden, um externe Geräte zugleich mit dem Computer betriebsbereit zu schalten.

Test-Eingang
Wenn man diesen Anschluß erdet, nehmen alle Adressen-, Daten- und Kontrollsignal-Puffer einen hochohmigen Zustand ein, so daß die zugehörigen Leitungen gewissermaßen abgeschaltet sind. Betroffen sind neben den Anschlüssen A0 – A15 und D0 – D7 die Signale WR', RD', IN', OUT', RAS', CAS' und MUX'. Dies dient normalerweise nur zu Testzwecken.

Warte-Eingang
Wenn man diesen Anschluß (Signal: WAIT') auf Erdpotential legt, wird die Funktion der Zentraleinheit unterbrochen. Dies ist sehr nützlich, wenn man ein Gerät mit dem Computer betreiben will, das langsamer arbeitet als der Rechner. Durch Schalten des WAIT'-Signals kann man das Gerät dann jeweils gegenüber dem Computer aufholen lassen.

Speicher-Schreib-Taktsignal
Dieses Signal (WR') kennzeichnet einen Einschreibvorgang der Computer-Zentraleinheit in den Speicher.

Speicher-Lese-Taktsignal
Dieses Signal (RD') zeigt an, daß die CPU gerade Information aus dem Speicher liest.

Peripherie-Schreib-Taktsignal
Wenn dieses Signal (OUT') auf niedrige Spannung schaltet (logisch "0"), soll die auf den Datenbus vorhandene Information von einem externen Empfänger (Anschlußgerät) übernommen werden, dessen Adresse auf den unteren 8 Adressenleitungen A0 bis A7 zu finden ist. (Die Anzahl der möglichen Adressen für Anschlußgeräte – d. h. Ausgänge – beträgt damit $2^8 = 256$).

Peripherie-Lese-Taktsignal
Wenn dieses Signal (IN') einen niedrigen Spannungszustand einnimmt, erwartet der Computer von einem Anschlußgerät über den Datenbus Informationen, daß er mit einer Adresse auf den ersten 8 Leitungen des Adreßbusses (A0 – A7) anspricht.

Interrupt-Eingang
Wenn dieser Eingang (INT') auf 0 gelegt wird, springt die Zentraleinheit in ihrer Befehlsverarbeitung zu einer vorbestimmten Adresse im Festwertspeicher (ROM). Dabei gibt es drei verschiedene Möglichkeiten: Bei den ersten führt die CPU diesen Sprung nicht aus, bei der zweiten geht der Sprung zur Adresse 0038 Hex. Die dritte Möglichkeit besteht darin, den Sprung zu einer beliebigen, vorher angegebenen Zieladresse durchführen zu lassen. Doch läßt sich dies beim TRS-80 infolge interner Schaltungen nicht verwirklichen.

Interrupt-Bestätigung
Dieses Signal (INTAK') geht in den niedrigen Zustand, wenn die CPU den Interrupt-Zustand einnimmt. Für externe Schaltungen kann INTAK' zur Bestätigung eines eingeleiteten Interrupt-Vorganges dienen.

5 Volt-Ausgang
Dieser Anschluß ist mit der 5-V-Stromversorgung des Computers verbunden. Allerdings empfiehlt es sich nicht, Zusatzgeräte damit zu betreiben, da die Leistung der Stromversorgung ziemlich genau auf den Bedarf des TRS-80 zugeschnitten ist. Außerdem liegt dieser Anschluß bei den mit Level II BASIC ausgerüsteten Computern auf Erdpotential.

Verbindungen zur Außenwelt
Der Busanschluß unter der Klappe links hinten am Computergehäuse ist zwar vom Hersteller für die Verbindung mit dem Erweiterungs-Bauteil vorgesehen. Es bietet dem TRS-80-Freund aber auch die Möglichkeit, selbst Zusatzgeräte mit dem Computer zu steuern. Dies können, wie bereits erwähnt, Drucker und ähnliches sein, aber auch Steuerungen für beliebige Maschinen, Elektrogeräte, Lampen usw. Wie sich diese Möglichkeit grundsätzlich nutzen läßt, soll hier kurz dargestellt werden:

Prinzipiell gibt es zwei Wege, Informationen aus dem Computer nach außen (und von draußen herein) zu tragen. Man kann externe Schaltungen wie Speicherbereiche behandeln und sie gewissermaßen zu einem Teil des Computers machen, oder man kann die beim Microprocessor Z-80 gegebene Möglichkeit nutzen, Signale an Ausgänge zu senden und von Eingängen zu empfangen. Im ersten Fall gibt es auch bei einem TRS-80 mit 16K freiem Speicherraum noch 32768 verschiedene Adressen, die man extern nutzen kann, da die CPU insgesamt 64K Speicherkapazität oder 65536 Speicherstellen ansprechen kann. Und in jede Speicherstelle läßt sich ein Byte einschreiben — das sind die Zahlen von 0 bis 255. Insgesamt also fast unerschöpfliche Möglichkeiten zur Steuerung oder zum Informationsaustausch mit der Außenwelt! Allerdings müssen Zusatzgeräte, die auf dieser Basis funktionieren, neben den acht Datenleitungen an 16 Adressenleitungen angeschlossen sein und deren Signale verarbeiten — unter Umständen ein nicht kleiner Aufwand.

Etwas einfacher geht es mit der anderen Variante, den (sogenannten) Ein- und Ausgängen: Hier sind außer den Datenleitungen nur acht Adressenleitungen zu berücksichtigen, und zwar die Leitungen A0 bis A7. Damit gibt es aber auch nur 255 verschiedene Ein- und Ausgänge (den 256. braucht der Computer, wie schon erläutert, selber). Allerdings wird das für die meisten Zwecke mehr als genug sein.

Für jede der beiden Methoden, Informationen auszutauschen, gibt es zwei Steuerleitungen: Für die Speicheransprache WR' und RD', für die Ein- und Ausgänge OUT' und INT' (wie bereits dargelegt).

Alle vier Signale werden vom Computer kontrolliert! Er entscheidet also, wann er Speicherplätze bzw. Anschlüsse ansprechen will und wann nicht. Darauf müssen etwaige Zusatzschaltungen reagieren und dürfen nicht von sich aus Signale in die Datenleitungen einspeisen.

Bei allen Zusatzschaltungen sollten zwei Regeln unbedingt beachtet werden: Immer eine eigene Stromversorgung vorsehen. Die Versorgung des TRS-80 reicht nur für den Computer selbst und kaum für weitere Geräte. Und was noch wichtiger ist: Daten-, Steuer- und Adressenleitungen sind nur für eine TTL-Last ausgelegt. In der Regel ist also stets zusätzliche Pufferung erforderlich.

An dieser Stelle soll auch noch auf eine weitere Möglichkeit hingewiesen werden, den Computer mit der Außenwelt zu verbinden und zwar mit Hilfe der Steuerleitung TEST'. Legt man dieses Signal, das sich am Anschluß Nr. 23 des Busanschlusses befindet, auf Nullpotential, gehen unter anderem alle Daten- und Adressenleitungen sowie alle Speicher-Steuerleitungen in den hochohmigen Zustand über. Damit ist die CPU von ihrem Speicher vollständig abgeschnitten. Man könnte also einen zweiten Computer Zugang zu dem Speicherraum verschaffen. Nun wird sich vielleicht kaum jemand zwei Computer hinstellen und betreiben. Doch die Möglichkeit dazu wäre gegeben: Mit vergleichsweise einfachen Zusatzschaltungen könnten zwei (oder vielleicht noch mehr?) TRS-80 zusammenarbeiten und gemeinsam ein wesentlich leistungsfähigeres Datenverarbeitungssystem bilden, bei dem zum Beispiel eine Seite zeitraubende Berechnungen übernimmt, während die andere für die interaktive Zusammenarbeit mit dem Menschen bereitsteht.

Nachzutragen bleiben noch die Verbindungen zum BASIC: Speicherplätze, ob im Computer oder außerhalb, werden in bekannter Weise mit PEEK und POKE angesprochen, Ein- und Ausgänge mit INP und OUT. Und noch eine Bemerkung: Es könnte sich, wählt man die Speicher-Variante für Anschlußgeräte, notwendig erweisen, diesen Bereich vor dem unkontrollierten Zugriff des Computers zu schützen. Dazu müßte man beim Anschalten auf die Frage MEMORY SIZE? mit der höchsten, im Computer vorhandenen Speicherstelle antworten (20479 bei 4K, 32767 bei 16K).

Erweiterung des RAM-Bereichs im TRS-80
Nicht wenige TRS-80-Freunde werden sich ihren Computer zunächst in der Version mit 4K freier Speicherkapazität gekauft haben bzw. kaufen wollen – mit der Absicht, zu einem späteren Zeitpunkt die Erweiterung auf 16K vorzunehmen, die ja nicht teurer ist, als den Computer gleich mit dem größeren Arbeitsspeicher (RAM) zu kaufen. Hier soll nun gezeigt werden, wie man die Speichererweiterung selbst vornehmen kann. So läßt sich unter günstigen Umständen einiges Geld sparen, und man behält die kleineren Speicherchips, die sonst beim Händler bleiben.

Gebraucht werden zunächst acht dynamische RAM-Chips vom Typ D416 oder 4116. Ferner ist etwas Schaltdraht für Brückenverbindungen erforderlich sowie ein Kreuzkopf-Schraubenzieher und ein Abzieher für integrierte Schaltungen.

Zunächst müssen alle Kabelverbindungen vom Computergehäuse entfernt werden. Dann legt man es mit der Tastatur nach unten auf eine weiche Unterlage (Handtuch, Kissen oder etwas ähnliches) und öffnet es, indem man alle sechs sichtbaren Schrauben entfernt. (Achtung: Es gibt drei Schraubengrößen!)

Dann sucht man sich aus den auf der Platine vorhandenen integrierten Schaltungen die acht RAMS Z13 bis einschließlich Z20 heraus (siehe Skizze). Sehr wichtig ist die Position der Kerben an den Gehäusen der Chips. Die Kerben in den neuen Chip-Gehäusen müssen unbedingt nach der Montage in die jeweils gleiche Richtung zeigen!

Die MOS-Chips, mit denen man es hier zu tun hat, können durch statische Entladungen, die vom menschlichen Körper ausgeht, leicht Schaden nehmen. Deshalb sollte man Schuhe und Strümpfe ausziehen, ehe man mit ihnen umgeht. Günstig ist es auch, sich vorher kurz an einem Heizkörper oder etwas ähnlichem zu erden. Anschließend sollte man unnötige Bewegungen vermeiden, um erneute statische Aufladung zu vermeiden.

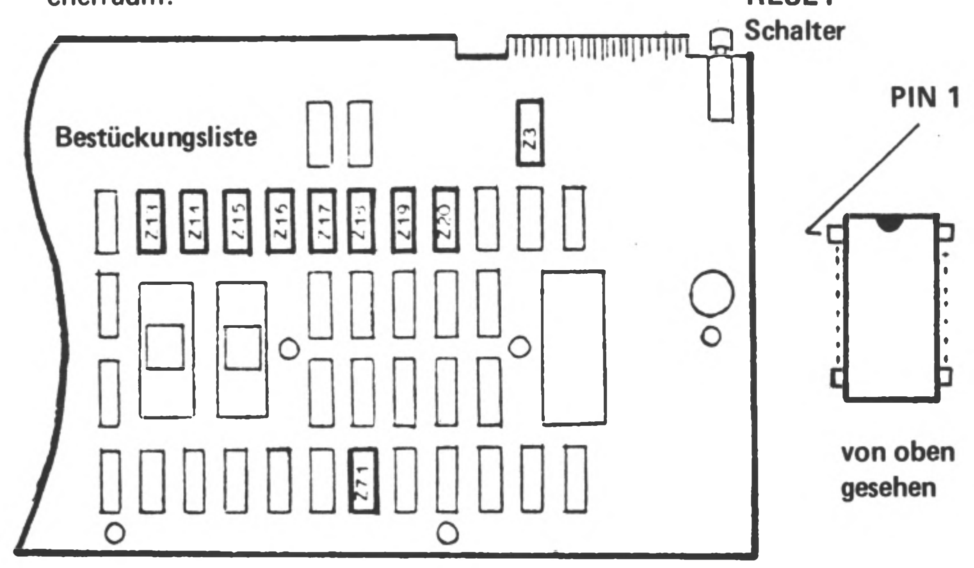
Jetzt kann man die alten integrierten Schaltungen mit dem Abziehwerkzeug oder (wenn es sich nicht vermeiden läßt) einem kleinen Schraubenzieher entfernen. Sie stecken alle in Sockeln. Dazu lockert

man sie vorsichtig an beiden Enden und legt sie zum Schutz gegen statische Elektrizität auf Aluminiumfolie oder leitendem Schaumstoff ab. An ihre Stelle kommen jetzt die neuen RAMs, wobei man besonders Acht geben muß, die Kontakte nicht zu verbiegen.

Jetzt muß man noch die Sockel Z3 und Z71 heraussuchen (s. Skizze) und die Drahtbrücken entfernen. Dann werden neue Drahtbrücken gemäß der Skizze eingesetzt (wenn man sich den Luxus leisten will, kann man auch DIP-Schalter verwenden). Ein wichtiger Hinweis: Es gibt zwei verschiedene Platinen-Typen mit den Bezeichnungen 170069A und 170069D. Die erste Version verwendet Brückensockel mit 14 Kontakten, die zweite mit 16 Kontakten (Skizze).

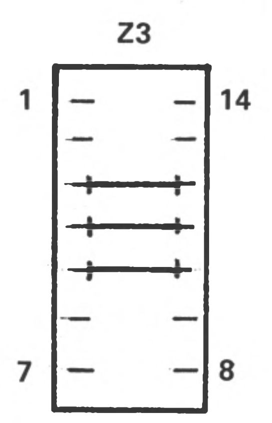
Man sollte sich genau vergewissern, mit welcher Platinenversion man es zu tun hat, und die Brücken entsprechend anordnen.

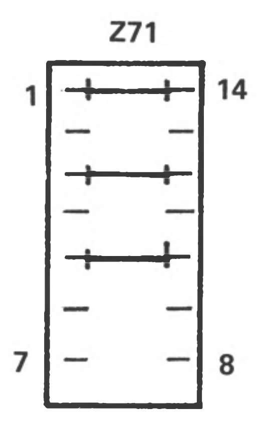
Nach einer sorgfältigen Überprüfung der Positionen von RAMs und Brückenverbindungen kann man jetzt die Gehäuse wieder zusammenbauen. Dabei sollte auf korrekten Sitz der Tastatur geachtet werden. Nach Anschluß der verschiedenen Kabel bleibt noch das Gerät anzuschalten und PRINT NEM einzutippen. Nach dem ENTER sollte die Zahl 15572 erscheinen. Und nun viel Vergnügen mit dem 16K Speicherraum!



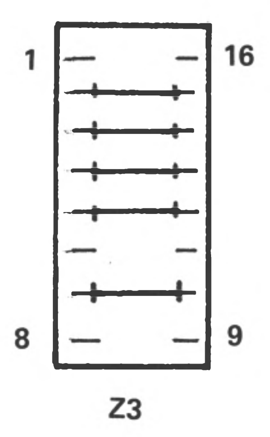
RAM- und Brückenpositionen

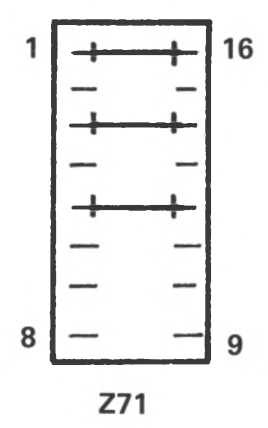
170069A Platine





170069D Platine





Brückenbeschaltung

Modifizierung des READ-Vorgangs

Es gibt beim TRS-80 einen sehr praktischen Trick, die READ-Anweisung abzuwandeln, die bekanntlich Daten ins eigentliche Programm übernimmt, die unter dem Begriff DATA zusammengefaßt stehen. Und zwar geht das durch Veränderung des Inhalts der Adresse 16553 des Computerspeichers. Wie aus dem Handbuch hervorgeht, befindet sich die Adresse in einem Speicherbereich, der zwar frei zum Einlesen von Informationen ist (RAM-Speicherbereich), aber für interne Zwecke vorbehalten bleibt. Normalerweise befindet sich in der Position 16553, die Zahl 255. Ersetzt man sie durch den Befehl POKE 16553,0, wird der Mechanismus ausgeschaltet, der normalerweise dafür sorgt, daß bei aufeinanderfolgenden READ-Befehlen jeweils auch aufeinanderfolgende Daten bzw. Strings in den DATA-Zeilen angesprochen werden. Folgendes kleine Programm soll dies verdeutlichen:

```
10  FOR I=1 TO 5
20  READ A
30  PRINT A;
40  NEXT
50  POKE 16553,0
60  FOR I=1 TO 5
70  READ A
80  PRINT A;
90  NEXT
100 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Läßt man das Programm laufen, erhält man folgende Bildschirmanzeige:

```
1 2 3 4 5 6 6 6 6 6
```

Nach Lesen und Ausgeben der ersten fünf Zahlen wurde das Weiterschalten auf die jeweils nächste Zahl unterbrochen und in der zweiten Programmschleife (Zeilen 60 bis 90) las der Computer immer wieder dieselbe Zahl. Beim zweiten und allen folgenden Läufen ergibt sich das folgende Bild:

```
1 1 1 1 1 1 1 1 1 1
```

Das kommt daher, daß die Wirkung des POKE-Befehls unabhängig von irgendwelchen Programmen immer bestehen bleibt. In diesem Falle ist der Weiterschalt-Mechanismus ausgeschaltet, und es wird immer

nur die erste Zahl, nämlich die 1 gelesen und auf den Bildschirm gebracht. Das zeigt auch, wie wichtig es ist, die Veränderung am Betriebssystem des Computers, die mit POKE 16553,0 geschah, nach Gebrauch wieder rückgängig zu machen – durch POKE 16553,255.

Zusammen sind die beiden Befehle POKE 16553,0 und POKE 16553,255 also ein recht praktisches Mittel, um die READ-Routine nach Bedarf abzuändern: Man kann durch die erste Anweisung das Weiterschalten anhalten und eine bestimmte Zahl (oder String) beliebig oft lesen, ehe man durch die zweite Anweisung zu den folgenden Zahlen weitergeht. Wenn man also eine Zahl oftmals benötigt, muß man nur die entsprechenden READ-Befehle zwischen den POKE-Anweisungen anordnen und braucht die Zahl in der DATA-Zeile nur einmal hinzuschreiben. Übrigens: Da das Weiterschalten (naturgemäß) jeweils nach dem READ-Vorgang geschieht, wird die Zahl, die mehrmals gelesen werden soll, auch nach Wiederinkraftsetzen der Weiterschaltung durch POKE 16553,255 noch einmal gelesen. Man muß also zwischen die beiden POKE-Befehle immer ein READ weniger anordnen, als man die betreffende Zahl tatsächlich braucht. Ein Beispiel:

```
10   READ A(0)
20   READ A(1)
30   POKE 16553,0
40   FOR I=2 TO 8
50   READ A(I)
60   NEXT
70   POKE 16553,255
80   READ A(9)
90   READ A(10)
100  FOR I=0 TO 10: PRINT A(I); : NEXT: END
110  DATA  2, 3, 999, 3
```

Ein Programmlauf gibt dann folgendes Bild:

2 3 999 999 999 999 999 999 999 999 3

Die Zahl 999 wird innerhalb der Schleife siebenmal gelesen und anschließend nach dem POKE 16553,255 noch einmal.

Ein kleiner Trick, der das Schreiben von Programmen erleichtert: Man kann an das Ende eines noch nicht fertigen Programms oder an einer vorübergehend gewählten Unterbrechungsstelle den Befehl LIST einfügen (eventuell auch mit Angabe der gewünschten Zeilen). Dann wird der Programmtext nach jedem Versuchslauf automatisch gelistet und man hat ihn gleich für Änderungen vor Augen. Achtung: LIST stoppt neben seiner eigentlichen Aufgabe die Programmausführung endgültig. Man kann ein Programm also nicht einfach mit CONT fortsetzen, wie bei STOP oder END.

Programme

Nichts geht über die Praxis. Diese Regel gilt natürlich auch für die Beschäftigung mit Microcomputern und hier insbesonders mit dem TRS-80. Deshalb ist auch ein guter Teil dieses Buches der Praxis gewidmet, nämlich den Programmen selbst. Davon finden sich hier, auf diesen Text folgend, eine ganze Anzahl und zwar mit ganz unterschiedlichen Längen und Schwierigkeitsgraden. Dem Zweck dieses Buches entsprechend sind sie für Level II BASIC geschrieben. Eine Reihe von Programmen läßt sich in 4K Speicherplatz unterbringen, der Rest erfordert 16K. In der Mehrzahl handelt es sich um Spiele, doch es sind auch einige praktische, für persönliche Finanzangelegenheiten darunter. Andere stellen Hilfen für den Computerbetrieb dar, auch ein paar nütziiche Routinen zum Einbau in eigene Programme des TRS-80-Freundes sind darunter.

Und damit ist auch gleich ein ganz wichtiges Thema angesprochen: Die Arbeit mit den hier abgedruckten Programmen: Empfehlenswert ist es, zunächst mit einfachen, kurzen Programmtexten zu beginnen und sich jede Zeile nach dem Eintippen genau anzusehen, ehe man mit der nächsten beginnt. Wenn man dann etwas Routine gewonnen hat, kann man sich an längere Programme heranwagen.

Bekanntlich kann man auf Leerstellen zwischen den einzelnen Ausdrücken des BASIC-Textes verzichten. Trotzdem empfiehlt es sich, sie wie in den meisten Programmen dieses Buches geschehen, einzufügen. Dadurch gewinnt der Text sehr an Lesbarkeit und erfahrungsgemäß ist dies gerade bei Programmen, die man nicht selbst geschrieben hat, besonders wichtig. Bei ihnen sind Fehler beim Eintippen ungleich schwerer zu finden.

Die einzelnen Programme des Buches stammen von unterschiedlichen Quellen. Dementsprechend unterscheiden sie sich zunächst in einzelnen Schreibweisen. Darüber an anderer Stelle noch näheres. Darüberhinaus erkennt man aber auch, daß bestimmte Aufgaben auf verschiedene Weise gelöst sind. Dies sollte nicht als Nachteil gesehen werden – im Gegenteil: Für den Leser bietet dies die Möglichkeit, die unterschiedlichen Methoden zum Programmieren einzelner Aufgaben kennenzulernen. Ihre Anwendung hängt von den Umständen ab, ist aber in manchen Fällen auch einfach Geschmackssache.

Überhaupt sollten die hier abgedruckten Programme nicht als „Heiligtümer" angesehen werden. Immer sind sie auch als Anregung gedacht – für Veränderungen nach dem Geschmack und den Bedürfnissen des Lesers. Sicher werden dem einen oder anderen auch einige Verbesserungen einfallen. Allerdings soll in diesem Zusammenhang darauf hingewiesen werden, daß manche der Programme mit Absicht nicht den höchstmöglichen Stand an Eleganz und Raffinesse darstellen. Dies ergibt sich zwangsläufig aus dem Bestreben, einigermaßen durchschaubare Programme zu schreiben und den Leser mit möglichst vielen Möglichkeiten bekanntzumachen, die im BASIC – insbesondere dem Level II BASIC von Radio Shack – stecken.

Noch eine Bemerkung zu den praktischen Programmen für persönliche Finanzen: Hier geht es „um Geld", das heißt, Fehler können sehr viel schwerwiegendere Folgen haben, als bei den Spielprogrammen. Daher empfiehlt es sich, die Ergebnisse von Berechnungen, die man mit ihnen macht, am Anfang sorgfältig zu prüfen. Erst wenn man ganz sicher ist, daß die richtigen Zahlen herauskommen, sollte man beginnen, sich auf sie zu verlassen.

Programmauswahl

Programmauswahl
Mit diesem kleinen Programm kann man vorteilhaft eine Programmcassette beginnen lassen. Es macht sich die Tatsache zunutze, daß als Programmbezeichnung in der Ladeanweisung auch eine String-Variable zulässig ist – CLOAD A$ statt wie üblich CLOAD "X". Auch kann man ja das CLOAD auch innerhalb eines Programms benutzen. Wenn also die Cassette mit diesem Programm beginnt, braucht man sich um den Inhalt des Bandes weiter keine Gedanken zu machen. Man legt es einfach ein und lädt das erste Programm – nämlich dieses. Dann kann man unter den gezeigten Möglichkeiten wählen, indem man den betreffenden Buchstaben eingibt. Den Rest – Suche und Laden des gewünschten Programms – besorgt der Computer dann von selbst.

```
10 CLS
20 PRINT "AUF DIESEM BAND SIND FOLGENDE PROGRAMME:
30 PRINT:PRINT
40 PRINT "a  -  (ERSTES PROGRAMM)
50 PRINT
60 PRINT "B  -  (ZWEITES PROGRAMM)
70 PRINT
80 PRINT "C  -  (DRITTES PROGRAMM)
90 PRINT
100 PRINT "D  -  (VIERTES PROGRAMM)
110 PRINT
120 PRINT "UM DAS GEWUENSCHTE PROGRAMM ZU LADEN,
                                    DEN ZUGEHOERIGEN
 130 INPUT "BUCHSTABEN EINTIPPEN UND 'ENTER'
                                   DRUECKEN - "; X$

 140 CLOAD X$
```

Die Türme von Hanoi

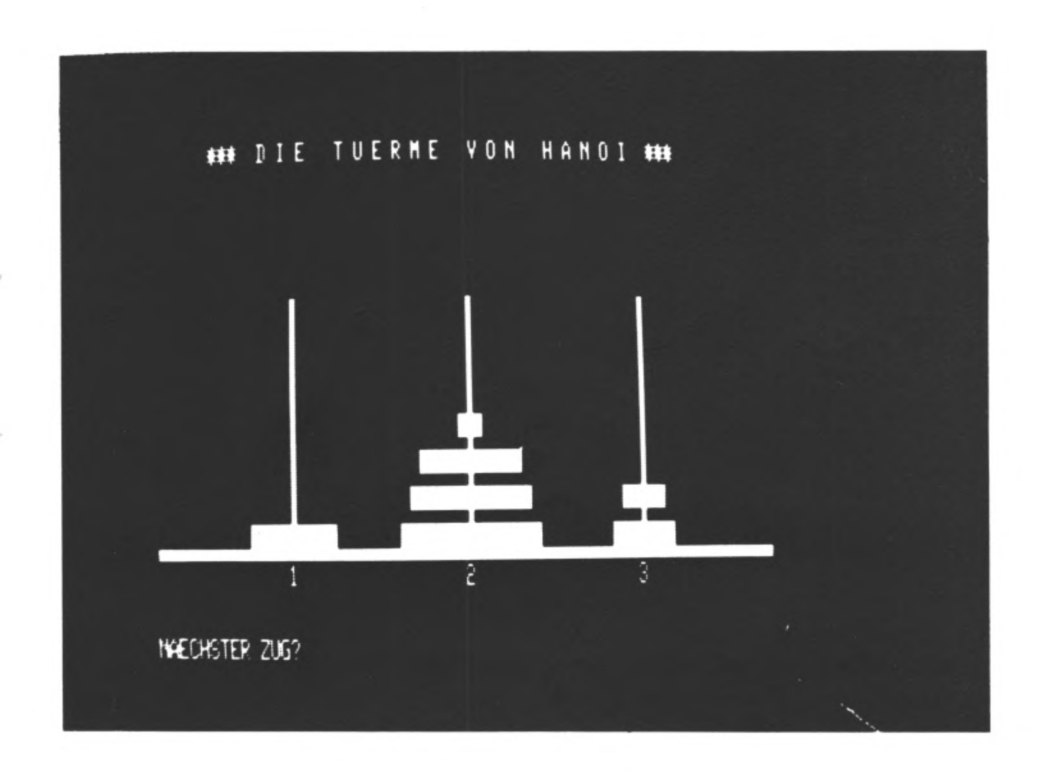
Die Türme von Hanoi
Dieses Spiel werden viele in ihrer Jugend gespielt haben, wenn auch nicht mit einem Computer. Auch unter dem Namen „Türme von Hanoi" ist es hierzulande wohl nicht unbedingt bekannt. Worum es geht, ist wieder in den Erläuterungen erklärt: Man hat drei Stangen, auf denen sieben runde Scheiben mit einem Loch in der Mitte stecken. Die Scheiben haben alle eine unterschiedliche Größe und das Spiel besteht darin, sie alle einzeln von einer Stange auf eine der beiden anderen zu heben, ohne daß jemals eine größere auf einer kleineren liegt. Der Computer prüft die einzelnen Bewegungen der Scheiben und weist unerlaubte Züge zurück. Auch zählt er die Züge und teilt ihre Gesamtzahl mit, wenn es dem Spieler gelungen ist, alle Scheiben von einer Stange auf eine andere zu bringen. Anschließend kann man weiterspielen oder neu beginnen.

Interessant übrigens die Art der Dateneingabe zum Bewegen der Scheiben: Die Stangen sind nummeriert und für eine Bewegung gibt man nur Nummern der Stange, von der abgehoben wird und der Zielstange ein, ohne ENTER drücken zu müssen. Dazu wird eine INKEY$-Routine zweimal hintereinander durchlaufen, die in den Zeilen 1010 bis 1030 steht und die eingetippten Zahlen werden auch auf dem Bildschirm angezeigt, als ob sie durch ENTER eingegeben worden wären. Das USING „ # #" in Zeile 1030 dient dabei nur dazu, die beiden Ziffern auf dem Bildschirm ein wenig näher aneinander rücken zu lassen. Insgesamt erspart diese Eingabe-Routine das Komma zwischen den Zahlen und das ENTER-Drücken und beschleunigt so das Spiel nicht unerheblich.

Zur Verdeutlichung sind die einzelnen Abschnitte des Programms mit entsprechenden Bemerkungen gekennzeichnet.

```
1 ' COPYRIGHT 1978, THE BOTTOM SHELF, INC.
2 ' ALL RIGHTS RESERVED.  DO NOT COPY
3 ' P.O. BOX 49104  ATLANTA GA 30359
40 CLS: PRINT"     *** D I E   T U E R M E   V O N   H A N
O I  ***
50 PRINT: PRINT: PRINT"  ZIEL DIESES SPIELS IST ES, ALLE SIE
BEN SCHEIBEN VON EINER
60 PRINT"STANGE ZU EINER ANDEREN ZU VERSETZEN. DIE SCHEIBEN
DUERFEN
70 PRINT"DABEI NUR EINZELN BEWEGT WERDEN, UND ES DARF NIEMAL
S EINE
80 PRINT"GROESSERE AUF EINER KLEINEREN LIEGEN. EINEN ZUG MAC
HT MAN,
90 PRINT"INDEM MAN NACHEINANDER DIE NUMMERN DER BEIDEN BETRE
FFENDEN
92 PRINT"STANGEN EINTIPPT.
94 PRINT: INPUT"ZUM STARTEN 'ENTER' DRUECKEN   ";A$
100 CLEAR 50  :  DEFINT A-Z  :  DIM A(3,7)
104 'SPIELANFANG
110 CLS: PRINT"     *** D I E   T U E R M E   V O N   H A N
 O I  ***
120 FOR X=1 TO 126: SET (X,35): NEXT
130 FOR Y=34 TO 14 STEP -1: FOR X=27 TO 99 STEP 36
140 SET (X,Y): NEXT X,Y
150 FOR Z=33 TO 15 STEP -3:FOR Y=Z+1 TO Z STEP -1
160 W=2*Z/3-8: FOR X=63-W TO 63+W: SET(X,Y) : NEXT X,Y,Z
170 FOR X=1 TO 3: PRINT@ 762+X*18, X;: NEXT X
180 FOR Y=1 TO 7: A(2,Y)=8-Y: NEXT Y
190 A(1,0)=0: A(2,0)=7: A(3,0)=0: N=0
200 PRINT@ 896,"NAECHSTER ZUG?";
204 ' ZUG ANNEHMEN
210 GOSUB 1000: F=A: GOSUB 1000: T=A
220 IF A(F,0) > 0 AND A(F,A(F,0)) < A(T,A(T,0)) THEN 250
225 IF A(T,0)=0 THEN 250
230 PRINT@ 919, "UNERLAUBTER ZUG - NOCH EINMAL, BITTE";
240 PRINT@ 910, "";: GOTO 210
250 PRINT@ 910, CHR$(30);: PRINT@ 910, "";
252 'EINE SCHEIBE BEWEGEN
255 Z=36-3*A(F,0): V=36*F-9: W=2*A(F,A(F,0))
260 FOR Y=Z TO Z+1: FOR X=V-W TO V+W
270 RESET (X,Y): NEXT X: SET (V,Y): NEXT Y
280 A(T,0)=A(T,0)+1: A(T,A(T,0))=A(F,A(F,0))
290 A(F,A(F,0))=0: A(F,0)=A(F,0)-1: N=N+1
```

```
300 Z=36-3*A(T,0): V=36*T-9: W=2*A(T,A(T,0))
310 FOR Y=Z+1 TO Z STEP -1: FOR X=V-W TO V+W
320 SET (X,Y): NEXT X,Y
330 IF A(1,0)<>7 AND A(3,0)<>7 THEN 210
334 ' SPIELENDE
340 PRINT@ 896,"GRATULIERE - DU HAST" N "ZUEGE GEBRAUCHT
350 PRINT"MOECHTEST DU WEITERMACHEN (J=JA, N=NEIN)?";
360 A$=INKEY$: IF A$<>"J" AND A$<>"N" THEN 360
365 PRINT@ 960, CHR$(30);
370 IF A$ = "J" THEN PRINT@ 896, CHR$(30);: GOTO 200
380 PRINT@ 960, "  NOCH EIN SPIEL (J/N) ?";
390 A$=INKEY$: IF A$="J" THEN 110 ELSE IF A$<>"N" THEN 390
400 CLS: END
1000 'ZUG ANNEHMEN
1010 A$=INKEY$: IF A$="" THEN 1000 ELSE A=VAL(A$)
1020 IF A<1 OR A>3 THEN 1010
1030 PRINT USING "##"; A;: RETURN
```



Dateneingabe mit DATA

Dateneingabe mit DATA
Nicht selten findet man BASIC-Programme, bei denen Daten nicht mit Hilfe der INPUT-Anweisung vom Benutzer angefordert werden, sondern als DATA-Zeilen eingegeben werden müssen und dann mit dem READ-Befehl in den Programmablauf übernommen werden. Dieses etwas umständlichere und auf den ersten Blick wenig sinnvolle Verfahren hat jedoch ganz bestimmte Vorteile: Will man die Daten für spätere Verwendung speichern, kann man das tun, indem man einfach das Programm noch einmal auf eine Cassette überspielt. Beim Wiedereinlesen zu einem späteren Zeitpunkt sind dann die Daten auch gleich wieder da. So erspart man sich sowohl Datenspeichern mit PRINT#-1, als auch das Einlesen mit INPUT#-1, beides bekanntlich besonders bei größeren Datenmengen außerordentlich zeitraubende Prozeduren.

Zu empfehlen ist die Dateneingabe durch Schreiben neuer DATA-Zeilen außerdem, wenn Veränderungen am Programm absehbar sind und man die eingegebenen Werte dabei nicht verlieren will. Auch kann der Fall eintreten, daß bestimmte Werte bei einem CLEAR-Befehl nicht verlorengehen sollen, aber trotzdem vor dem Start des Programmes noch nicht festgelegt sein dürfen.

Dieses Programm soll als Beispiel dafür dienen, wie die Dateneingabe mit DATA-Zeilen, die ja nicht während des eigentlichen Programmablaufs möglich ist, trotzdem für den BASIC-Unkundigen Computerbenutzer einigermaßen verständlich erfolgen kann. Selbstverständlich sind auch andere Methoden möglich, die herauszufinden aber der Phantasie des Lesers überlassen bleiben soll.

Die Programmroutine ist geeignet zur Eingabe einer beliebigen Anzahl von Daten zwischen 1 und 100. Festgelegt ist dies in Zeile 30, mit der DIM-Anweisung, die sich natürlich den jeweiligen Bedürfnissen anpassen läßt. Nur wenn noch keine DATA-Zeilen vorhanden sind, tritt die Da-

teneingabe in Aktion. Das besorgt die Fehler-Routine ON ERROR GOTO 1000 zusammen mit der Zeile 35. Führt das READ A(0) zu einem Fehler, was ja ohne DATA-Zeilen passiert, springt der Computer zur Zeile 80, das heißt zur Erläuterung der Dateneingabe. Andernfalls fährt der Computer mit Zeile 40 fort. Hier befinden sich die READ-Befehle für die übrigen Daten als Programmschleife. Wenn alle vorhandenen Daten übernommen sind und somit hier ein Fehler auftritt, greift wieder die Fehler-Routine ein und bricht die READ-Schleife ab.

Die Zeile 500 schließlich ist stellvertretend für ein beliebiges Programm eingebaut, das mit der Dateneingabe nach diesem Muster arbeiten soll. Dabei ist „X–1" die Zahl der Daten. Der Wert stammt aus der Zeile 40, wo die Schleife nach Übernahme aller vorhandenen Daten abgebrochen wurde. Die erste Zahl wurde allerdings bereits durch Zeile 35 gelesen; daher beginnt die Schleife in Zeile 500 mit Y=0 statt mit Y=1.

```
5 ON ERROR GOTO 1000
10 CLS
30 DIM A(100)
35 READ A(0)
40 FOR X=1 TO 100: READ A(X): NEXT
80 PRINT "DATENEINGABE - ZUERST 'DATA' EINTIPPEN, DANN VORGESEHENE DATEN
90 PRINT "DURCH KOMMAS GETRENNT EINGEBEN. JEWEILS SPAETESTENS NACH
100 PRINT "RD. 3 1/2 ZEILEN 'ENTER' DRUECKEN UND DANACH WIEDER
110 PRINT "MIT DATA BEGINNEN.
120 PRINT "NACH ENDE DER DATENEINGABE DIE 'BREAK'-TASTE
130 PRINT "DRUECKEN UND PROGRAMM NEU STARTEN.
140 INPUT "ALLES KLAR? DANN 'ENTER' DRUECKEN."; X
150 CLS
160 AUTO 200,1
500 FOR Y=0 TO X-1: PRINT A(Y),: NEXT
510 END
1000 IF ERL=35 RESUME 80 ELSE IF ERL=40 RESUME 500
```

Ballistik

Ballistik
Dieses Programm stellt ein einfaches, aber recht reizvolles Bildschirmspiel dar. Von links nach rechts fliegen in unterschiedlicher Höhe Pfeile über den Bildschirm. Der Spielende muß versuchen, sie mit einer Kanone abzuschießen, die in der rechten unteren Ecke des Bildschirms zu denken ist. Die Geschosse fliegen dabei auf echten ballistischen Bahnen. Den Abschußwinkel kann man beim Abschuß durch Wahl der dabei verwendeten Zifferntaste bestimmen. Den Ziffern 0 bis 9 entsprechen dabei die Winkel 0^0 (waagerecht) bis 90^0 (senkrecht).

Der Wert R in Zeile 3 ist die Anfangsgeschwindigkeit des Geschosses. Er ist so gewählt, daß bei einem Abschußwinkel von 45^0 die waagerechte und senkrechte Geschwindigkeitskomponenten jeweils 1 wären. Natürlich kann man das Geschoss durch eine andere Wahl des R auch beliebig schneller oder langsamer machen. Die Wirkung der Schwerkraft auf das Geschoss spiegelt sich in dem Wert von I (Zeile 10) wieder. Auch hier kann man nach Belieben Änderungen vornehmen. Allerdings empfiehlt es sich nicht, das Geschoss wesentlich schneller zu machen, als im Programm vorgesehen. Dann wird nämlich auf dem Bildschirm häufiger eine Druckposition durch den Lichtpunkt übersprungen und die Testroutine, die einen Treffer feststellen soll (Zeile 130), funktioniert in manchen Fällen nicht mehr.

```
1 ON ERROR GOTO 1000
2 CLS: PRINT "                    ***** BALLISTIK *****": PRINT: P
RINT:PRINT: PRINT "ZUM SCHIESSEN UND FUER SCHUSSWINKEL ZIFFE
RN 0 - 9 DRUECKEN.": FOR Q=1 TO 1000: NEXT
3 PA=.1745329: R=SQR(2)
5 CLS: FOR P=1 TO 500: NEXT
10 X=125: Y=47: I=.02: N%=0: A$=""
12 Z=64*(RND(15)-1)
15 FOR K=0 TO 60
40 IF A$<>"" THEN 100
60 A$=INKEY$: IF A$="" FOR J=1 TO 10: NEXT J: GOTO 140
80 A=VAL(A$): H=R*SIN(PA*A): W=R*COS(PA*A)
100 RESET(X,Y): X=X-2*W: Y=Y-H+N%*I: SET(X,Y)
120 N%=N%+1
130 IF PEEK(15363+K+Z)<>32 AND PEEK(15363+K+Z)<>128 THEN 600
140 PRINT @ K+Z, " -->";
200 NEXT K: PRINT @ 61+Z, "   ";: GOTO 12
600 FOR T=1 TO 5: PRINT @ K+Z, "****";: FOR U=1 TO 50: CMD U
: PRINT @ K+Z, "    ";: FOR U=1 TO 50: NEXT U,T
620 GOTO 5
1000 A$="": X=125: Y=47: N%=0
1010 RESUME 140
```

Labyrinth

Labyrinth
Dies ist ein Spiel, das der Computer im wesentlichen mit sich selbst spielt. Dem TRS-80-Freund bleibt die Rolle eines Zuschauers, was in diesem Falle aber faszinierend genug ist. Das Programm läuft in zwei Phasen ab: Als ersten Schritt zeichnet es einen Irrgarten auf den Bildschirm und zwar in einem Raster von 4 x 4 Bildpunkten. Es handelt sich um ein völlig zufälliges Labyrinth, das jedesmal neu und anders gezeichnet wird. Links und rechts hat es eine Öffnung im begrenzenden Rand, und vom Eingang zum Ausgang gibt es immer nur einen einzigen Weg. Aber es gibt ihn auch immer.

Während der Irrgarten entsteht, und auch nachdem er fertig ist, hat der Bediener eine Eingriffsmöglichkeit. Durch Drücken entsprechender Tasten kann wahlweise die zweite Programmphase eingeleitet oder das Irrgarten-Zeichnen neu gestartet werden.

Die zweite Programmphase besteht darin, daß sich ein Lichtpunkt selbständig den Weg von links nach rechts durch das Labyrinth sucht. Dabei findet er stets den richtigen Weg und probiert jede Möglichkeit garantiert nur einmal.

Die Tatsache, daß ein Irrgarten immer mit einem, aber nur einem Durchgangsweg gezeichnet wird, ist das Resultat der Programmzeilen 1050 und 1230 bis 1260. Die erste sorgt dafür, daß eine neu zu zeichnende Irrgarten-Mauer immer an einer bereits bestehenden beginnt, die anderen bewirken, daß eine Mauer stets ohne weitere Berührung mit einer anderen endet. Beim Zeichnen beginnen die neuen Mauern also irgendwo an einer bestehenden und pflanzen sich dann zufällig fort, bis es nicht mehr weitergeht, ohne wieder auf eine Mauer zu stoßen. Dann wiederholt sich der Vorgang. Siebenmal wird in zufälliger Weise eine neue Richtung getestet, wenn es in einer bestimmten

nicht mehr weitergeht. Dann geht das Programm dazu über, einen Ansatzpunkt für eine neue Mauer zu suchen. Dies spiegelt sich in Zeile 1100 wieder (IF U=7 ...). Durch Ändern dieser Zahl kann man wahlweise Labyrinthe mit durchschnittlich kürzeren oder längeren Mauern erzielen.

Gleichzeitig ergibt sich ein Einfluß auf die Zeit, die das Irrgarten-Zeichnen braucht: Die letzten Ansatzpunkte für Mauern, die noch bestehende Lücken füllen, findet der Computer meist erst nach längerem Suchen, weil er dabei auch nach dem Zufallsprinzip vorgeht. Daher also: Je kürzere Mauern, desto mehr Ansatzpunkte und damit auch desto längere Labyrinth-Zeichenzeit. Andererseits kostet es natürlich auch Zeit, den Computer unnötig lange nach nicht vorhandenen Möglichkeiten zur Verlängerung der Mauern suchen zu lassen, an denen er gerade baut.

Die Tatsache, daß der Computer oft recht lange braucht, um die letzten Lücken im Irrgarten zu füllen, ist auch der Grund dafür, daß das Zeichnen abgebrochen werden kann, um zur zweiten Programmphase überzugehen. Die letzten, noch fehlenden kurzen Mauern verändern das Labyrinth nicht mehr wesentlich und der Bediener soll nicht die Geduld verlieren.

Im zweiten Programmteil werden zunächst alle offenen und versperrten Durchgänge auf zwei zweidimensionale Matrizen übertragen (S(30,11) und W(31,10) – Zeilen 2020 und 2070). Dann wird der Punkt am linken Eingang des Irrgartens auf die Reise geschickt. Der Lichtfleck bewegt sich schrittweise und kontrolliert nach jedem Schritt, in welche Richtungen er weitergehen kann. Erste Priorität genießt dabei eine Bewegung nach rechts, in Richtung Ausgang. Ist dies nicht möglich, oder kommt der Punkt gerade aus dieser Richtung, geht es nach oben oder unten weiter (wenn beides möglich ist, ist die Auswahl hier zufällig). Notfalls bewegt sich der Punkt auch rückwärts, d. h. nach links.

Mit Hilfe der Matrix P(30,10), die gerade soviel Elemente hat, wie es Positionen für den Punkt im Labyrinth gibt, wird festgelegt, wo der Punkt auf seiner Suche nach dem Weg durch den Irrgarten jeweils schon gewesen ist. Bei jedem Durchgang des Punktes durch eine Position erhöht sich der Wert des zugehörigen Elements der Matrix auf eine Kennzahl, die im Verlauf des Weges des Punktes immer weiter ansteigt. Durch Überprüfung dieser Zahl wird erreicht, daß jede Sackgasse nur einmal aufgesucht wird.

Hat der Lichtfleck den Ausgang erreicht, kann wieder ein neuer Durchgang gewählt werden. Da eine Auswahl der Richtungen „nach oben" und „nach unten" zufällig erfolgt, kann der zweite Weg durch denselben Irrgarten durchaus etwas anders aussehen als der erste, wenn der richtige Weg auch immer gleich bleibt. Oder es ist auch wieder der Neustart des Programmes möglich. Auf jeden Fall ist der Computer zunächst wieder in der Zeichenphase, so daß man ihn auch etwaige Lükken im Labyrinth füllen lassen kann.

Etwas durcheinanderbringen kann man die Suche des Lichtfleckes nach dem Ausgang, wenn man den Punkt schon kurz nach Beginn des Irrgarten-Zeichnens losschickt. Auf den dann vorhandenen, großen freien Flächen kann er sich zwar beliebig bewegen, kann aber zunächst nicht dahin, wo er schon einmal war. Das Resultat sind verwirrte, kreisende Bewegungen, ehe dann schließlich doch der Ausgang gefunden wird.

```
10 CLS: PRINT@ 516, CHR$(23)"D A S  L A B Y R I N T H
20 DEFINT A-Z: RANDOM
30 FOR I=0 TO 2000: NEXT
100 CLS: FOR X=0 TO 124: SET(X,0): SET(X,44): NEXT
110 FOR Y=1 TO 43: SET(0,Y): SET(124,Y): NEXT
120 FOR Y=21 TO 23: RESET(0,Y): RESET(124,Y): NEXT
900 PRINT@ 960, "n = NOCHMAL ANFANGEN, W = WEITER IM PROGRAM
M";
950 ON ERROR GOTO 10000: DIM P(30,10), W(31,10), S(30,11), Z
(3)
1000 U=0: A$=INKEY$: IF A$="N" RUN ELSE IF A$="W" THEN 2000
1020 X=RND(32)*4-4: Y=RND(12)*4-4
```

```
1050 IF NOT POINT(X,Y) THEN 1020
1100 IF U=7 THEN 1000 ELSE T=RND(4)-1
1200 FOR Z=0 TO 4
1220 ON T GOTO 1240, 1250, 1260
1230 IF POINT(X+4,Y) U=U+1: GOTO 1100 ELSE U=0: SET(X+Z,Y):
GOTO 1300
1240 IF POINT(X-4,Y) U=U+1: GOTO 1100 ELSE U=0: SET(X-Z,Y):
GOTO 1300
1250 IF POINT(X,Y+4) U=U+1: GOTO 1100 ELSE U=0: SET(X,Y+Z):
GOTO 1300
1260 IF POINT(X,Y-4) U=U+1: GOTO 1100 ELSE U=0: SET(X,Y-Z)
1300 NEXT
1400 IF T=0 X=X+4 ELSE IF T=1 X=X-4 ELSE IF T=2 Y=Y+4 ELSE I
F T=3 Y=Y-4
1500 GOTO 1100
2000 PRINT@ 960, "LOS GEHT'S..."TAB(45);
2020 FOR A=0 TO 31: FOR B=0 TO 10: W(A,B)=POINT(4*A,4*B+2):
NEXTB,A
2070 FOR A=0 TO 30: FOR B=0 TO 11: S(A,B)=POINT(4*A+2,4*B):
NEXTB,A
2500 X=2: Y=22: SET(X,Y): A=0: B=5: W(0,5)=-1: P(A,B)=1
3000 H=P(A,B): A=(X-2)/4: B=(Y-2)/4: FOR N=0 TO 3: Z(N)=255
3050 NEXT: IF A=30 AND B=5 K=0: GOTO 5000
3100 P(A,B)=H
3500 IF W(A+1,B) GOTO 3600
3550 Z(0)=P(A+1,B)
3600 IF S(A,B) GOTO 3700
3650 Z(1)=P(A,B-1)
3700 IF S(A,B+1) GOTO 3800
3750 Z(2)=P(A,B+1)
3800 IF W(A,B) GOTO 4000
3850 Z(3)=P(A-1,B)
4000 L=K:Q=255: FOR N=3 TO 0 STEP-1:IF Z(N)<=Q THEN Q=Z(N):
K=N
4050 NEXT: IF K=3-L THEN P(A,B)=P(A,B)+1
4100 IF K=1 AND Z(1)=Z(2) K=RND(2)
5000 FOR N=0 TO 3: RESET(X,Y): ON K GOTO 5100, 5200, 5300
5050 X=X+1: GOTO 5500
5100 Y=Y-1: GOTO 5500
5200 Y=Y+1: GOTO 5500
5300 X=X-1
5500 SET(X,Y): NEXT
```

```
6000 IF X=126 PRINT@ 960, "DURCHGEKOMMEN! ! !"TAB(45);: FOR
N=0 TO 2000: NEXT: CLEAR: RESET(126,22): GOTO 900
7000 GOTO 3000
10000 RESUME 1000
```

Wissenschaftliche Notation

Wissenschaftliche Notation
Dieses Programm macht mit der sogenannten „wissenschaftlichen" oder exponentiellen Schreibweise von Zahlen bekannt, die auch der Nichttechniker heutzutage oft vorgesetzt bekommt, etwa bei Taschenrechnern. Auch der private Computerfreund sollte mit solchen Zahlen umgehen können, da ja das BASIC (auch das vom TRS-80, natürlich) mit ihnen operiert. Es kann also nicht schaden, wenn man mit den Werten von der Form $a \cdot 10^b$ etwas anfangen kann, und dazu ist Übung in der Potenzrechnung nötig. Mit diesem Programm kann man die Übung erwerben: Es stellt dem Anwender immer neue, mit Hilfe von Zufallszahlen gebildete Rechenaufgaben, zählt die richtigen und falschen Antworten und bildet daraus einen Leistungsdurchschnitt. Dabei wird bei der Beurteilung der Grundzahl einigermaßen großzügig vorgegangen; meist genügt es, sie im Kopf überschlägig zu schätzen, um —zusammen mit dem korrekten Exponenten — zu einer richtigen Antwort zu kommen. Für die Anwendung ist wichtig, daß man Grundzahl und Exponent ohne weitere Zeichen getrennt eingeben muß. Um den Rest kümmert sich der Computer.

```
1 REM 'SCIENTIFIC NOTATION PRACTICE'
2 REM  (C) 1978  CLOAD MAGAZINE
3 CLS: CLEAR 6Ø: Y=1:N=Ø:INPUT"BRAUCHST DU ERLAEUTERUNGEN (W
ENN JA, 'JA' EINGEBEN)";A$
4 IF A$<>"JA" THEN 2Ø
5 CLS: PRINT"UM SEHR GROSSE ODER KLEINE ZAHLEN AUSZUDRUECKEN
, VERWENDET MAN
6 PRINT"DIE 'WISSENSCHAFTLICHE' ODER 'EXPONENTIELLE SCHREIBW
EISE.
7 PRINT"DIE ZAHLEN SEHEN DANN FOLGENDERMASSEN AUS:
8 PRINT"                                      P
9 PRINT"                              A  X  1Ø
1Ø PRINT"ZWEI SO GESCHRIEBENE ZAHLEN MULTIPLIZIERT MAN IN DI
ESER WEISE:
11 PRINT"          P                 Q
  (P  +  Q)
12 PRINT" A  X  1Ø    X    B  X  1Ø     =    (A  X  B)  X
1Ø
13 PRINT"UND SO DIVIDIERT MAN SIE:
14 PRINT"          P                 Q
  (P  -  Q)
15 PRINT" A  x  1Ø    /    B  X  1Ø     =    (A  /  B)  X
1Ø
16 PRINT"DAS HEISST, BEIM MULTIPLIZIEREN ADDIERT MAN DIE EXP
ONENTEN,
17 PRINT"             BEIM DIVIDIEREN SUBTRAHIERT MAN DIE EXP
ONENTEN.
18 PRINT"DAS WOLLEN WIR JETZT UEBEN.": INPUT"BIST DU BEREIT?
 DANN DRUECK 'ENTER'-";A$
19 W=Ø:R=Ø
2Ø CLS:S=RND(2)
3Ø ON S GOSUB1ØØØ,12ØØ
35 IF S=1 GOTO55
4Ø S=RND(2)
5Ø IF S=1 D=1 : G=Ø : GOTO8Ø
55 PRINT@ 133, STRING$(5Ø,45)
6Ø S=RND(2)
7Ø ON S GOSUB2ØØØ,22ØØ
8Ø N=N/D
82 F=F-G
1ØØ PRINT@516,"DEINE LOESUNG";
12Ø INPUTX
13Ø PRINT@538," X  1Ø"
```

```
14Ø PRINT@416,"";
15Ø INPUTK
155 IFN=Ø K=F
16Ø PRINT@635,"";
2ØØ IF (ABS(X-N)<.126)*(F=K) GOTO3ØØ
21Ø W=W+1:PRINT@ 641,"DAS IST LEIDER FALSCH.";
22Ø GOTO4ØØ
3ØØ R=R+1:PRINT@ 641,"DAS IST RICHTIG ! ! !";
4ØØ PRINT@ 769,"DIE GENAUE LOESUNG LAUTET:";
41Ø PRINT@865,F;
42Ø PRINT@911,N;"  X   1Ø";
43Ø  IF F>Ø F=F-1:N=N*1Ø:GOTO43Ø
44Ø  IF F<Ø F=F+1:N=N/1Ø:GOTO44Ø
45Ø PRINT@94Ø,"ODER  "N;
5ØØ PRINT@96Ø,"";
5Ø5 PRINTW"FALSCH "R"RICHTIG  = ";
515 PRINT USING "###%";R*1ØØ/(W+R);: INPUT"   WEITER (<ENTER
> DRUECKEN)";A$
52Ø GOTO2Ø
1ØØØ A=RND(2Ø)-5
1Ø1Ø E=RND(2Ø)-1Ø
1Ø2Ø L=8Ø:GOSUB5ØØØ
1Ø4Ø N=A:F=E:RETURN
12ØØ A=RND(1Ø)-2
121Ø E=RND(1Ø)-5
122Ø L=7Ø:GOSUB5ØØØ
124Ø PRINT@ L+23,"X";
125Ø N=A:F=E
127Ø A=RND(12)-4
128Ø E=RND(1Ø)-5
129Ø L=98:GOSUB5ØØØ
13ØØ N=N*A:F=F+E:RETURN
2ØØØ A=RND(12)-4
2ØØ5 IFA=ØGOTO2ØØØ
2Ø1Ø E=RND(14)-7
2Ø2Ø L=272:GOSUB5ØØØ
2Ø4Ø D=A:G=E:RETURN
22ØØ A=RND(1Ø)-2
22Ø5 IFA=ØGOTO22ØØ
221Ø E=RND(12)-6
222Ø L=262:GOSUB5ØØØ
224Ø PRINT@ L+23,"X";
225Ø D=A:G=E
```

```
2270 A=RND(8)
2280 E=RND(9)-4
2290 L=290:GOSUB5000
2300 D=D*A:G=G+E:RETURN
5000 PRINT@ L,"(";A;" X  10          )";
5030 PRINT@ L-52,E;
5040 RETURN
```

Biorhythmen

Biorhythmen

Wer sich für Biorhythmen interessiert, die es ermöglichen sollen, den körperlichen, geistigen und seelischen Zustand eines Menschen für jeden Tag seines Lebens festzustellen, für den wird dieses Programm sicher sehr nützlich sein. An einzugebenden Daten verlangt es zuerst Geburtsdatum der Person, dessen Biorhythmen man wünscht.
Dann folgt die Frage nach dem Monat, für den die drei Rhythmus-Kurven errechnet werden sollen. Unter geschickter Ausnutzung der graphischen Fähigkeiten des TRS-80 werden dann drei sinus-ähnliche Kurven auf den Bildschirm gemalt – für den angegebenen Berechnungsmonat.

Anschließend kann man entweder Geburtsdatum oder Berechnungsmonat für einen neuen Durchlauf ändern (oder beides), worauf der Computer drei neue Kurven mit den geänderten Daten zeichnet. Kurze Erläuterungen finden sich als Bemerkungen am Anfang des Programmes, so daß man sie sich durch Auflisten (LIST) auf den Bildschirm holen kann. Wem diese Lösung nicht gefällt, kann selbstverständlich auch etwas anderes einbauen, zum Beispiel ein Ausdrucken des Textes vor dem eigentlichen Programmstart.

Auf falsche Eingaben reagiert das Programm mit der Antwort „unzulässiges Datum!", und dabei werden alle Schaltjahr-Möglichkeiten genau beachtet. Nur einen Fehler verkraftet es nicht, nämlich solche, bei nachträglichen Änderungen des Geburtstages (also, wenn bereits ein- oder mehrmals die Kurven gezeichnet wurden), die zur Folge haben, daß das Geburtsdatum später liegt als der Berechnungsmonat. Dann kommt der Computer aus der Programmroutine zur Zurückweisung falscher Eingaben nicht mehr heraus, und man muß neu beginnen. Vielleicht wäre es für den Leser eine reizvolle Aufgabe, das Programm so zu verändern, daß auch dieser Fehler richtig berücksichtigt wird.

Auf bemerkenswerte Weise wird übrigens erreicht, daß links und rechts neben der Graphik mit den Biorhythmen-Kurven senkrechte Schriftzüge stehen. Dafür werden am Anfang des Programms zwei String-Variable definiert, die nichts weiter enthalten als die Anweisungen "eine Stelle rückwärts" und "neue Zeile und eine Stelle rückwärts". Die betreffenden Wörter werden dann so auf den Bildschirm gebracht, daß zwischen jedem Buchstaben eine solche Anweisung steht. Auf der rechten Seite muß man dabei "eine Stelle rückwärts" verwenden, weil das "neue Zeile" ja am Ende jeder Zeile auf dem Bildschirm bereits automatisch erfolgt.

```
3 '      * BIORHYTHMEN *
4 ' 2. KURVE IST EMOTIONAL, 3. KURVE INTELLEKTUELL
5 ' ZUM NEUSTART DES pROGRAMMS 1 EINGEBEN, ZUR AENDERUNG DES
6 ' BERECHNUNGSMONATS 2 UND ZUR AENDERUNG DES
7 ' GEBURTSDATUMS 3 EINGEBEN.
8 X=1
9 A$=CHR$(24): B$=CHR$(26)+CHR$(24)
10 CLS
20 PRINT@279,"BIORHYTHMEN
24 PRINT@464,"DEIN GEBURTSTAG:
28 PRINT@527," ";
30 INPUT"TAG,  MONAT,  JAHR";K,J,L
70 E=J:F=K:G=L
90 Z=1
100 GOSUB220
103 IFZ>1500THEN10
105 GOSUB350
110 D=A
111 ONXGOTO112,112,120,112
112 CLS
113 PRINT@278,"BIORHYTHMEN";
114 PRINT@396," ";
115 PRINT"ZAHL DES MONATS UND DES JAHRES FUER";
116 PRINT@459," ";
117 PRINT"DIE GEWUENSCHTEN BIORHYTHMEN EINGEBEN";
118 PRINT@530," ";
119 INPUT"MONAT,  JAHR ";M,O
120 E=M:F=1:G=O
140 Z=1
150 GOSUB220
```

```
153 IFZ>15ØØTHEN114
155 GOSUB35Ø
16Ø D=A-D
165 IFD>-KTHEN17Ø
166 GOSUB24Ø
167 GOTO7Ø
17Ø GOTO1ØØØ
22Ø IF(G<Ø)+(G>4ØØØ)THEN24Ø
225 IF(E<1)+(E>12)THEN24Ø
23Ø ONEGOTO26Ø,28Ø,26Ø,34Ø,26Ø,34Ø,26Ø,26Ø,34Ø,26Ø,34Ø,26Ø
24Ø CLS
244 PRINT@47Ø,"UNZULAESSIGES DATUM!
246 FORZ=1TO15ØØ:NEXT
25Ø RETURN
26Ø IFF>31THEN24Ø
27Ø RETURN
28Ø IFG/4ØØ=INT(G/4ØØ)THEN32Ø
29Ø IFG/1ØØ=INT(G/1ØØ)THEN31Ø
3ØØ IFG/4=INT(G/4)THEN32Ø
31Ø IFF>28THEN24ØELSE33Ø
32Ø IFF>29THEN24Ø
33Ø RETURN
34Ø IFF>3ØTHEN24Ø
35Ø DATAØ,31,59,9Ø,12Ø,151,181,212,243,273,3Ø4,334
355 DATA-1
36Ø RESTORE
365 FORZ=1TOE
37Ø READA
375 NEXTZ
376 GOSUB15ØØ
377 Z=1
38Ø A=A+G*365+INT(G/4)+F+1-INT(G/1ØØ)+INT(G/4ØØ)
39Ø IFINT(G/4)=G/4THEN41Ø
4ØØ GOTO45Ø
41Ø IFG/4ØØ=INT(G/4ØØ)THEN43Ø
42Ø IFG/1ØØ=INT(G/1ØØ)THEN44Ø
43Ø IFE>2THEN45Ø
44Ø A=A-1
45Ø RETURN
1ØØØ P=D-INT(D/23)*23
1Ø1Ø I=D-INT(D/28)*28
1Ø2Ø E=D-INT(D/33)*33
1Ø3Ø CLS
```

```
1035 GOSUB18ØØ
1Ø4Ø Y=23
1Ø5Ø FORX=6TO122STEP4
1Ø6Ø SET(X,Y-2)
1Ø7Ø NEXTX
1Ø75 IFD<ØTHEN1115
1Ø8Ø IFP=ØTHEN112Ø
1Ø9Ø FORA=1TOP
11ØØ READZ
111Ø NEXTA
1115 Q=D
112Ø FORX=3TO123STEP4
1122 IFQ>=ØTHEN113Ø
1124 Q=Q+1
1126 NEXTX
113Ø READZ
114Ø FORY=ZTO43
115Ø SET(X,Y-2)
116Ø NEXTY
117Ø NEXTX
118Ø GOSUB15ØØ
1185 IFD<ØTHEN1225
119Ø IFI=ØTHEN123Ø
12ØØ FORA=1TOI
121Ø READZ
122Ø NEXTA
1225 Q=D
123Ø FORX=4TO124STEP4
1232 IFQ>=ØTHEN124Ø
1234 Q=Q+1
1236 NEXTX
124Ø READZ
125Ø FORY=3TO44
1255 SET(X,Y-3)
126Ø NEXTY
127Ø RESET(X,Z-2)
128Ø NEXTX
129Ø GOSUB15ØØ
1295 IFD<ØTHEN1335
13ØØ IFE=ØTHEN134Ø
131Ø FORA=1TOE
132Ø READZ
133Ø NEXTA
```

```
1335 Q=D
1340 FORX=5TO125STEP4
1342 IFQ>=ØTHEN135Ø
1344 Q=Q+1
1346 NEXTX
135Ø READZ
137Ø SET(X,Z-2)
139Ø NEXTX
14ØØ GOTO17ØØ
15ØØ READZ
151Ø IFZ=-1THENRETURN
152Ø GOTO15ØØ
17ØØ INPUTX
171Ø ONXGOTO1Ø,172Ø,1Ø,1Ø
172Ø CLS:GOTO7Ø
18ØØ IF(M=1)+(M=3)+(M=5)+(M=7)+(M=8)+(M=1Ø)+(M=12)THEN2ØØØ
181Ø IFM=2THEN1825
182Ø PRINT@897," 1   3   5   7   9  11  13  15  17  19  21
23";
1821 PRINT"  25  27  29   1";:GOTO2Ø2Ø
1825 IFO/4ØØ=INT(O/4ØØ)THEN185Ø
183Ø IFO/1ØØ=INT(O/1ØØ)THEN184Ø
1835 IFO/4=INT(O/4)THEN185Ø
184Ø PRINT@897," 1   3   5   7   9  11  13  15  17  19  21
23";
1841 PRINT"  25  27   1   3";
1842 PRINT@961,"   2   4   6   8  1Ø  12  14  16  18  2Ø  22
";
1843 PRINT"  24  26  28   2";
1844 GOTO2Ø4Ø
185Ø PRINT@897," 1   3   5   7   9  11  13  15  17  19  21
23";
1851 PRINT"  25  27  29   2";
1852 PRINT@961,"   2   4   6   8  1Ø  12  14  16  18  2Ø  22
";
1853 PRINT"  24  26  28   1";
1854 GOTO2Ø4Ø
2ØØØ PRINT@897," 1   3   5   7   9  11  13  15  17  19  21
23";
2Ø1Ø PRINT"  25  27  29  31";
2Ø2Ø PRINT@961,"   2   4   6   8  1Ø  12  14  16  18  2Ø  22
";
2Ø3Ø PRINT"  24  26  28  3Ø";
```

```
2040 Y=43
2050 FORX=8TO121STEP8
2055 SET(X,Y-1)
2060 SET(X,Y):SET(X,Y+1)
2070 NEXTX
2075 PRINT@63,"*"A$"B"A$"I"A$"O"A$"R"A$"H"A$"Y"A$"T"A$"H"A$"
M"A$"E"A$"N"A$"*";
2100 ONMGOTO2102,2112,2122,2132,2142,2152
2101 ONM-6GOTO2162,2172,2182,2192,2202,2210
2102 PRINT@64,"J"B$"A"B$"N"B$"U"B$"A"B$"R";:GOTO2250
2112 PRINT@64,"F"B$"E"B$"B"B$"R"B$"U"B$"A"B$"R";:GOTO2250
2122 PRINT@192,"M"B$"A"B$"E"B$"R"B$"Z";:GOTO2250
2132 PRINT@192,"A"B$"P"B$"R"B$"I"B$"L";:GOTO2250
2142 PRINT@256,"M"B$"A"B$"I";:GOTO2250
2152 PRINT@192,"J"B$"U"B$"N"B$"I";:GOTO2250
2162 PRINT@192,"J"B$"U"B$"L"B$"I";:GOTO2250
2172 PRINT@128,"A"B$"U"B$"G"B$"U"B$"S"B$"T";:GOTO2250
2182 PRINT@64,"S"B$"E"B$"P"B$"T"B$"E"B$"M"B$"B"B$"E"B$"R";:G
OTO2250
2192 PRINT@64,"O"B$"K"B$"T"B$"O"B$"B"B$"E"B$"R";:GOTO2250
2202 PRINT@64,"N"B$"O"B$"V"B$"E"B$"M"B$"B"B$"E"B$"R";:GOTO22
50
2210 PRINT@64,"D"B$"E"B$"Z"B$"E"B$"M"B$"B"B$"E"B$"R";
2250 V=INT(O/10):W=INT(V/10):R=INT(W/10)
2255 U=O-V*10:T=V-W*10:S=W-R*10
2260 W=R:V=640:GOSUB2600
2270 W=S:V=704:GOSUB2600
2280 W=T:V=768:GOSUB2600
2290 W=U:V=832:GOSUB2600
2300 PRINT@895," ";
2310 RESET(0,44)
2500 RETURN
2600 ONW+1GOTO2605,2615,2625,2635,2645,2655,2665,2675,2685,2
690
2605 PRINT@V,"0";:RETURN
2615 PRINT@V,"1";:RETURN
2625 PRINT@V,"2";:RETURN
2635 PRINT@V,"3";:RETURN
2645 PRINT@V,"4";:RETURN
2655 PRINT@V,"5";:RETURN
2665 PRINT@V,"6";:RETURN
2675 PRINT@V,"7";:RETURN
2685 PRINT@V,"8";:RETURN
```

```
2690 PRINT@V,"9";:RETURN
9000 DATA 23,17,13,9,6,4,3,5,7,11,15,21,25,31,35,39,41,43,42
,40,37,33,29
9001 DATA 23,17,13,9,6,4,3,5,7,11,15,21,25,31,35,39,41,43,42
,40,37,33,29
9002 DATA 23,17,13,9,6,4,3,5,7,11,15,21,25,31,35,39,41,43,42
,40,37,33,29
9003 DATA -1
9010 DATA 23,19,15,11,7,5,4,3,4,5,7,11,15,19,23,27,31,35,39,
41,42,43,42
9011 DATA 41,39,35,31,27,23,19,15,11,7,5,4,3,4,5,7,11,15,19,
23,27,31,35
9012 DATA 39,41,42,43,42,41,39,35,31,27,23,19,15,11,7,5,4,3,
4,5,7,11,15
9013 DATA -1
9020 DATA 23,19,15,12,9,7,5,4,3,3,4,6,8,11,14,17,21,25,28,32
,35,38,40,42
9021 DATA 43,43,42,41,39,37,34,30,27,23,19,15,12,9,7,5,4,3,3
,4,6,8,11,14
9022 DATA 17,21,25,28,32,35,38,40,42,43,43,42,41,39,37,34,30
,27
9023 DATA -1
```

Raumjäger

Raumjäger

Dies ist ein weiteres Programm, das ein Bildschirmspiel erzeugt, bei dem der Computerfreund sich vorstellen soll, ein Kampfraumschiff zu steuern und feindliche Raumschiffe abzuschießen. Im Gegensatz zu dem Programm Stingray ist hier allerdings noch zusätzlich das gezielte Abfeuern der Strahlenwaffe möglich. Abgesehen davon ist dieses Programm dem Stingray sehr ähnlich.

```
1 'COPYRIGHT 1978 THE BOTTOM SHELF INC. / 1979 STUEBS
2 'ALL RIGHTS RESERVED  DO NOT COPY
3 'P.O. BOX 49104  ATLANTA GA 30359
4 CLS:GOSUB500
10 ON ERROR GOTO 2000
20 TX=RND(100)+9:TY=RND(30)+3
25 CLS:Q1=RND(120):N1=0
30 PRINT@0,NR"FEINDLICHE RAUMSCHIFFE ZERSTOERT, "NW"ENTKOMME
N
40 IFNW+NR=>20GOTO1500
100 PRINT@530,"<<<<<";:PRINT@539,"+";:PRINT@544,">>>>>";
105 IFA$<>" "THEN275
110 PRINT@980,"=";:PRINT@994,"=";
120 PRINT@917,"=";:PRINT@929,"=";
130 PRINT@854,"=";:PRINT@864,"=";
140 PRINT@791,"=";:PRINT@799,"=";
150 PRINT@728,"=";:PRINT@734,"=";
160 PRINT@665,"=";:PRINT@669,"=";
170 PRINT@602,"=";:PRINT@604,"=";
180 PRINT@539,"=";
```

```
190 PRINT@980," ";:PRINT@994," ";
200 PRINT@917," ";:PRINT@929," ";
210 PRINT@854," ";:PRINT@864," ";
220 PRINT@791," ";:PRINT@799," ";
225 PRINT@728," ";:PRINT@734," ";
230 PRINT@667," ";:PRINT@665," ";:PRINT@669," ";
240 PRINT@603," ";:PRINT@602," ";:PRINT@604," ";
250 PRINT@539," ";
255 IF TX>51 AND TX<59 AND TY>22 AND TY<27 THEN 400
275 PRINT@530,"     ";:PRINT@544,"     "
300 N1=N1+1:IFN1=>Q1THENNW=NW+1:GOTO20
305 X=0:Y=0
307 A$=INKEY$
310 IF A$="J" X=-RND(15)
315 IF A$="K" X=RND(15)
320 IF A$="U" Y=-RND(5)
325 IF A$="M" Y=RND(5)
330 IF A$="Z" GOSUB500:GOTO20
332 SX=RND(7)-4:SY=RND(5)-3
335 RESET(TX,TY):RESET(TX-1,TY+1):RESET(TX+1,TY+1)
340 TX=TX+SX+X:TY=TY+SY+Y
345 SET(TX,TY):SET(TX-1,TY+1):SET(TX+1,TY+1)
360 IF A$=" "THEN110ELSE100
400 FORXX=1TO10:PRINT@538,"***":GOSUB1005:PRINT@538,"   ":GO
SUB1005:NEXTXX:NR=NR+1
405 PRINT@473,"* * *":PRINT@537,"*  *  *":PRINT@601,"* * *
410 PRINT@408,"*  *  *":PRINT@534,"*    *    *":PRINT@664,"*
  *  *
415 PRINT@341,"*     *     *":PRINT@530,"*        *        *"
:PRINT:PRINT@725,"*     *     *
420 FORX=0TO500:NEXT:GOTO20
500 PRINT@20,"RAUMJAEGER"
510 PRINT"DU SITZT AM STEUER EINES RAUMJAEGERS UND HAST DIE
AUFGABE,
520 PRINT"FEINDLICHE RAUMSCHIFFE ABZUSCHIESSEN. SIE FUEHREN
530 PRINT"AUSWEICHMANOEVER AUS UND VERSUCHEN, IHREN RAKETENM
OTOR
540 PRINT"ZU ZUENDEN UND ZU ENTKOMMEN.
560 PRINT:PRINT"DEIN JAEGER LAESST SICH WIE FOLGT STEUERN:
570 PRINT"  J - NACH LINKS  K - NACH RECHTS
580 PRINT"  U - AUFWAERTS   M - ABWAERTS
590 PRINT"  LEERTASTE: STRAHLENKANONE ABFEUERN"
600 PRINT@896,"ZUM SPIELBEGINN IRGENDEINE TASTE DRUECKEN
```

```
604 PRINT" ('Z' RUFT DIE ERLAEUTERUNGEN WIEDER AUF)";
606 ED$=INKEY$:IF ED$=""GOTO606
610 CLS:RETURN
1005 FORX=1TO25:NEXTX:RETURN
1020 NW=NW+1:GOTO20
1500 IFNR<10PRINT:PRINT"DU MUSST NOCH UEBEN - "NW"SCHIFFE SI
ND ENTKOMMEN!":GOTO1520
1505 IFNR>=10 AND NR<15PRINT:PRINT"NICHT SCHLECHT, ABER"NW"S
CHIFFE SIND ENTKOMMEN.":GOTO1520
1510 IFNR>=15PRINT:PRINT"GUTER SCHUETZE! DU HAST"NR"ERWISCHT
.
1520 FORX=1TO1000:NEXTX:ED$=INKEY$:RUN
2000 RESUME 20
```

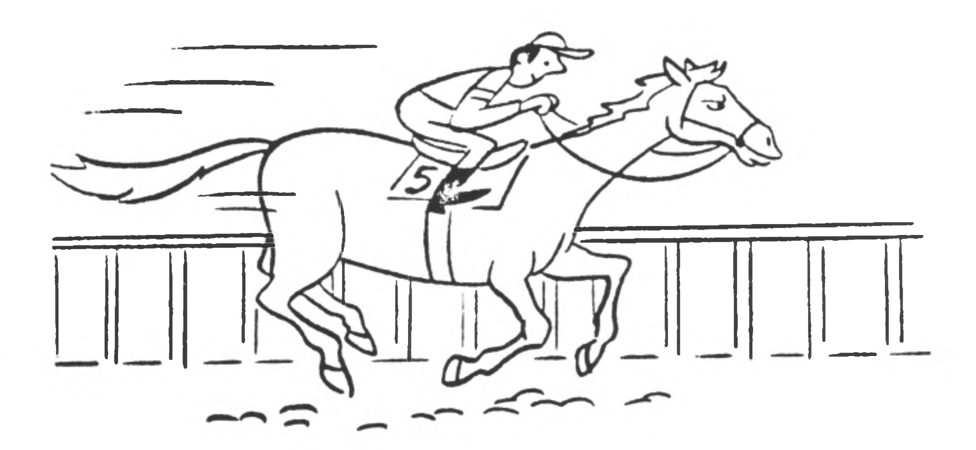
Buchstaben

hochschießen

Buchstaben hochschießen
Bei diesem Programm kann man eine beliebige Wortfolge (bis 150 Wörter bzw. 250 Buchstaben oder andere Zeichen) eingeben. Die durch Leerstellen getrennten Zeichengruppen (im allgemeinen Wörter) werden dann einzeln am unteren Rand des Bildschirms in Breitschrift abgebildet und die einzelnen Zeichen der Reihe nach aufwärts über den Bildschirm „geschossen". Eigentlich ist das Programm damit nicht besonders interessant — es eignet sich vielleicht für Werbezwecke oder ähnliches. Immerhin kann man daran aber einiges über die Verwendung der String-Funktion MID$ (A$,x,x) lernen. Zunächst muß die gesamte Wortfolge, die eingegeben wurde, in einzelne Wörter zerlegt werden. Das geschieht in den Zeilen 14170, 14200 und 14205. N1 ist am Ende dieses Vorgangs die Zahl der durch Leerstellen getrennten Zeichengruppen. Weiterhin muß jedes einzelne Wort wiederum in seine Buchstaben bzw. Zeichen zerlegt werden, damit diese einzeln „hochgeschossen" werden können. Das leistet die Zeile 14310 innerhalb der Schleife FOR W4=1 TO W1 ... NEXT W4. In beiden Fällen wird die genannte String-Funktion verwendet.

```
14000 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
14010 'ALL RIGHTS RESERVED   DO NOT COPY
14020 'p.o. BOX 49104  ATLANTA, GA.  30359
14025 CLS
14030 PRINT"DIESES PROGRAMM NIMMT EINE WORTFOLGE AN, BILDET
DIE EINZELNEN":PRINT"WOERTER AB UND SCHIESST DIE BUCHSTABEN
NACHEINANDER NACH OBEN.
14080 CLEAR1500:DIMB$(150)
14150 PRINT"DEINE NACHRICHT (BIS 150 WOERTER BZW. 250 STELLE
N):
14160 INPUTA$:CLS
14165 N1=1
14170 FORX=1TOLEN(A$):A1$=MID$(A$,X,1):A2$=MID$(A$,X+1,1)
14200 GOSUB14800
14205 IFA2$=" "THENN1=N1+1
14210 NEXTX
14300 FORX=1TON1
14309 W1=LEN(B$(X))
14310 FORW4=1TOW1:CLS:FORY=15TO1STEP-1:W2$=MID$(B$(X),W4,1)
14319 IFY<14THENPRINT@(Y*64)+80+(W4*2),CHR$(23)"   ";
14320 PRINT@980,CHR$(23)B$(X);
14321 PRINT@(Y*64)+(W4*2)+18,CHR$(23);W2$;
14325 NEXTY:NEXTW4
14330 NEXTX
14340 CLS:FORX=1TO300:NEXTX:GOTO14300
14800 B$(N1)=B$(N1)+A1$
14810 RETURN
```

Pferderennen



Pferderennen
Dieses Programm ist ein Wettspiel, an dem bis zu 10 Personen teilnehmen können. Es finden 10 Rennen statt, zu denen jeweils fünf Pferde antreten. Vor jedem Rennen kann jeder Teilnehmer auf eines der Pferde Geld setzen und erhält, falls sein Pferd gewinnt, die fünffache Summe gutgeschrieben. Andernfalls ist das Geld verloren. Jeder Spieler gibt zu Anfang des Spiels an, wieviel Geld er zum Verwetten hat, und der Computer rechnet gewonnene und verlorene Summen von diesem Betrag aus zu bzw. ab. Er überprüft auch, ob man mehr setzen will, als man hat oder man sein ganzes Geld verwettet hat. In diesem Fall scheidet der betreffende Spieler aus. Nach zehn Rennen, oder wenn alle Spieler ausgeschieden sind, ist das Spiel zu Ende. Gegen versehentliches Setzen auf ein nicht vorhandenes Pferd (z. B. ein Pferd Nr. 7) schützt das Programm nicht: Es soll keiner daran gehindert werden, Dummheiten zu begehen!

```
10000 'COPYRIGHT 1978  THE BOTTOM SHELF INC. / 1979 STUEBS
10010 'ALL RIGHTS RESERVED   DO NOT COPY
10020 'P.O. BOX 49104  ATLANTA GA 30359
10025 CLEAR750:DIM H(10),PL$(10),MN(10),HR(10),BT(10),P0(10)
10028 Z$=STRING$(50,45)
10030 CLS:PRINT@20,"PFERDERENNEN
10040 PRINT:PRINT"  WILLKOMMEN AUF DEM RENNPLATZ.  SIE WERDE
N JETZT AM SPORT
10050 PRINT"DER KOENIGE TEILNEHMEN.  DIE PFERDE STEHEN BEREI
T.
10060 PRINT:PRINT"  AUF ALLE PFERDE STEHEN DIE CHANCEN 5 ZU
1.
10070 INPUT"ZUM FORTFAHREN 'ENTER' DRUECKEN - ";AL
10080 CLS:PRINT@512,"";:INPUT "WIEVIELE SPIELER";PL
10085 IFPL<1ORPL>10PRINT:PRINT"NUR 1 BIS 10 SPIELER, BITTE!"
:FORX=1TO750:NEXTX:GOTO10080
10090 FORX=1TOPL
10100 CLS:PRINT@512,"";:PRINT"NAME DES SPIELERS NR."X;:INPUT
PL$(X)
10110 CLS:PRINT@512,PL$(X)", WIEVIEL GELD HAST DU ZUM WETTEN
";:INPUTMN(X):IFMN(X)<=0THEN10110
10120 NEXTX
10130 FORN3=1TO10
10135 FORX=1TOPL:IFMN(X)=0THEN10160
10140 CLS:PRINT@512,PL$(X)", WIEVIEL MOECHTEST DU IM RENNEN
NR."N3"SETZEN";:INPUTBT(X)
10145 IFBT(X)<0THEN10140ELSEIFBT(X)=0THEN10160
10148 IFBT(X)>MN(X)PRINT"SOVIEL HAST DU DOCH GAR NICHT!":FOR
Z=1TO500:NEXTZ:GOTO10140
10150 CLS:PRINT@512,PL$(X)", AUF WELCHES PFERD MOECHTEST DU
SETZEN (NR. 1 BIS 5)";:INPUTHR(X)
10160 NEXTX
10170 CLS
10900 PRINT"RENNEN NR."N3
10910 PRINT@25,"R E N N P L A T Z
10920 FORX=0TO10:PRINT@(X*64)+124,"!":NEXT
10930 FORX=0TO10STEP2:PRINT@(X*64)+74,Z$;:NEXT
10950 N5=0
11000 A2$="=="+CHR$(156)+CHR$(156)+CHR$(140)+CHR$(156)+CHR$(
174)+CHR$(131)
11010 N5=N5+1:IFN5>5THENWN=RND(5)
11015 IFN5<=5THENWN=WN+1
11020 IFH(WN)<10THENH(WN)=10:GOTO11029ELSEH(WN)=H(WN)+RND(3)
```

```
11Ø29 D2=(WN*128)+(H(WN))
11Ø3Ø PRINT@WN*128,STRING$(H(WN)," ");:PRINT@D2-4,WN;:PRINT@
D2,A2$;
11Ø4Ø IFH(WN)=>55THEN115ØØ
11Ø5Ø GOTO11Ø1Ø
11Ø55 IFH(1)=1ØFORY=1TO3ØØ:NEXTY
11Ø6Ø GOTO11Ø1Ø
115ØØ PRINT@896,"PFERD NR."WN"GEWINNT.
1151Ø PRINT"DIE GEWINNQUOTE BETRAEGT 5 ZU 1.";
1152Ø FORX=1TO6ØØ:NEXTX
1153Ø CLS
1154Ø FORX=1TOPL:IFP(X)=1THEN11584
1155Ø IFHR(X)=WNTHENMN(X)=MN(X)+(5*BT(X))
1156Ø IFHR(X)<>WNTHENMN(X)=MN(X)-BT(X)
1157Ø IFHR(X)=WN PRINTPL$(X)" HAT"5*BT(X)"DM GEWONNEN UND BE
SITZT JETZT"MN(X)"DM.
1158Ø IFHR(X)<>WN PRINTPL$(X)" HAT"BT(X)"DM VERLOREN UND BES
ITZT JETZT"MN(X)"DM.
11584 NEXTX
11585 PRINT:INPUT"ZUM WEITERSPIELEN 'ENTER' DRUECKEN - ";OP:
CLS
116ØØ FORX=1TOPL
116Ø5 P1=P1+1:IFP1<=P2THEN1162Ø
1161Ø IFP(X)=ØANDMN(X)=ØPRINTPL$(X)" IST PLEITE UND SCHEIDET
 FUER DIESEN RENNTAG AUS.":P(X)=1:PØ=1:PX=PX+1
1162Ø NEXTX
1163Ø IFPØ=1FORX=1TO1ØØØ:NEXTX:PØ=Ø
1197Ø WN=Ø:N5=Ø
1198Ø FORX=1TO5:H(X)=Ø:NEXTX
11985 IFPX=PLRUN
1199Ø NEXTN3
12ØØØ RUN
```

Reaktionstest

Reaktionstest

Dieses kleine Programm ist ganz reizvoll als Zeitvertreib unter Freunden, auf Parties usw. Vielleicht läßt sich auch der eine oder andere davon überzeugen, daß er zuviel getrunken hat und besser nicht den eigenen Wagen für die Heimfahrt nimmt, wenn er bei dem Test versagt. Interessant wird es auch dadurch, daß die Zeit, während das „Achtung", „Fertig" und „Los" vor dem Loslaufen des Pfeils erscheint, jeweils zufällig gewählt wird (Zeile 17200). Das PRINT@512," "; in Zeile 17060 soll nur die Schrift der INPUT-Anweisung an die gewünschte Stelle des Bildschirms bringen. Dabei sind die beiden Anführungszeichen eigentlich überflüssig und können weggelassen werden.

Hierzu einige grundsätzliche Bemerkungen: Wenn man sich die Programme ansieht, die für den TRS-80 erhältlich sind und aus den verschiedensten Quellen stammen können, wird man häufig feststellen, daß einiges Überflüssiges vorhanden ist oder gewisse Aufgaben umständlicher gelöst sind als insbesondere im Radio Shack Level II BASIC möglich. Dies kann (außer schlichten Fehlern) verschiedene Gründe haben.

Zum einen gibt es Programme, die mit geringfügigen Änderungen sowohl für Level I als auch für Level II BASIC funktionieren sollen. Natürlich müssen sie sich dann nach den beschränkteren Möglichkeiten des Level I richten. Weiterhin sind manche – vor allem einfache – Programme so geschrieben, daß sie auf (fast) jedem Microcomputer laufen, enthalten also z. B. keine Ausdrücke, die für den TRS-80 charakteristisch sind. Schließlich wurde auch das Radio Shack Level II BASIC weiter entwickelt, so daß manche Dinge heute einfacher zu machen sind als früher.

Auch in diesem Buch ist nicht jedes Programm bis zum letzten I-Tüpfelchen auf dem neuesten Stand der Entwicklung. Doch steht es dem Leser ja frei, für seinen persönlichen Gebrauch nach Belieben Änderungen vorzunehmen, wenn er sich daran stört. Außerdem wird jeweils auch deutlich, welche unterschiedlichen Ausdrucksformen im Programm möglich sind, ohne daß sich am Ende etwas ändert.

Ein Beispiel dafür findet sich in diesem Programm, wie oben bereits erwähnt. Die in diesem Fall überflüssigen Anführungszeichen wird man auch in anderen Programmen finden. Ein weiteres, auffallendes Beispiel ist im Programm „Startrek" enthalten.

```
17000 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
17010 'ALL RIGHTS RESERVED  DO NOT COPY
17020 'P.O.BOX  49104  ATLANTA GA  30359
17050 CLS:PRINT@17,CHR$(23)"REAKTIONSTEST
17055 FORX=0TO1000:NEXT
17060 CLS:CLEAR200:PRINT@512,"";:INPUT"WIE HEISST DU";NA$
17200 R1=RND(600):R2=RND(500):R3=RND(400)
17210 CLS:PRINT@532,"GLEICH GEHT ES LOS
17220 A$=CHR$(30)
17260 PRINT@0,"                  1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
17270 PRINT"1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
17280 PRINT@960,"WENN DER PFEIL ERSCHEINT, IRGENDEINE TASTE DRUECKEN";
17290 FORX=0TO500:NEXT
17300 FORX=1TOR1:NEXT:PRINT@512,A$TAB(20)"ACHTUNG....
17310 FORX=1TOR2:NEXT:PRINT@512,A$TAB(20)"FERTIG.....
17320 FORX=1TOR3:NEXT:PRINT@512,A$TAB(20)"LOS!":A1$=INKEY$
17330 IFN>1PRINT@128+N,"  [";
17335 N=N+2:E$=INKEY$:IFE$<>""THEN17500
```

```
17340 PRINT@128+N,"[";
17350 IFN=>62PRINT@512,A$TAB(20)"DU WARST ZU LANGSAM":FORX=1TO1000:NEXT:FORX=1TO
10:FORY=1TO35:NEXTY:CLS:FORY=1TO30:NEXTY:PRINT@512,CHR$(23)"DU WARST ZU LANGSAM
! ! !":NEXTX:GOTO17060
17360 GOTO17330
17500 FORX=1TO500:NEXT
17510 N1=INT(N/2)
17520 IFN1=<3THENT$="PHANTASTISCH!"
17530 IFN1>3ANDN1=<5THENT$="SUPERSCHNELL"
17540 IFN1>5ANDN1=<7THENT$="GUT"
17550 IFN1>7ANDN1=<9THENT$="MITTELMAESSIG"
17560 IFN1>9ANDN1=<11THENT$="UNTER DEM DURCHSCHNITT"
17570 IFN1>11ANDN1=<15THENT$="DU BRAUCHST UEBUNG"
17580 IFN1>15THENT$="SCHLECHT"
17600 FORX=1TO1000:NEXT:PRINT@320,NA$", DEINE REAKTIONSZEIT:":FORX=1TO10:FORY=1T
050:NEXTY:PRINT@512,T$A$:FORY=1TO40:NEXTY:PRINT@512,A$:NEXTX:GOTO17060
```

Die Springer-Tour

Die Springer-Tour
Dieses Programm bringt ein Spiel auf den TRS-80-Bildschirm, das einen amüsanten Zeitvertreib darstellt. Worum es sich dabei handelt, ist schon durch den erklärenden Text im Programm gesagt: Ein Problem mit einer Schachfigur – nämlich dem Springer – und dem Schachbrett. Bleibt eigentlich nur zu erwähnen, daß der Computer die Bewegungen der Figur verfolgt, falsche Züge nicht ausführt, die bereits besuchten Felder markiert und die Züge zählt. Für die, die es nicht wissen: Ein Springer darf bei einem Zug immer nur zwei Felder voran und eins zur Seite gehen.

```
10 DIMA(128)
100 GOSUB 5050: GOSUB 1500: GOSUB 2000: GOTO 3000
200 'DEN SPRINGER ZEICHNEN
250 FORI=1TO5
260 Z=V+I-1
270 IFI=1J=W:K=W+2
280 IFI=2J=W-1:K=W+3
290 IFI=3J=W+1:K=W+4
300 IFI=4J=W:K=W+3
310 IFI=5J=W-1:K=W+4
400 FORX=JTOK
410 IFS=1SET(X,Z):GOTO450
420 RESET(X,Z)
450 NEXTX,I
470 IFS=1SET(W-2,V+2):SET(W-1,V+2)
480 IFS<>1RESET(W-2,V+2):RESET(W-1,V+2)
500 RETURN
```

```
1500 'SPIEL-ANFANG
1515 S=2:O=Ø
1520 CLS:PRINT"LOS GEHT'S";
1530 FORI=1TO128:A(I)=Ø:NEXTI
1990 RETURN
2000 'SCHACHBRETT
2010 CLS:FORI=ØTO3
2030 FORJ=ØTO6ØSTEP2Ø
2040 FORV=6TO1Ø
2050 FORK=ØTO9
2060 SET(K+J,V+I*1Ø)
2070 SET(K+J+1Ø,V+5+I*1Ø)
2080 NEXTK,V,J,I
2150 FORI=1TO4
2155 J=43+128*I
2160 PRINT@J,I;
2165 PRINT@J+448,I+4;
2170 NEXTI
2200 IFO=ØGOTO28ØØ
2210 FORI=65TO128
2220 IFA(I)=ØGOTO25ØØ
2230 J=A(I)/8
2240 K=8*(J-INT(J))
2245 J=INT(J)+1
2250 IFK=ØK=8:J=J-1
2255 A=1
2256 IF(INT((J+K)/2)-(J+K)/2)=ØRESET(1Ø*K-9,5*J+5):RESET(1Ø*
K-8,5*J+5):A=Ø
2260 IFA=1SET(1Ø*K-9,5*J+5):SET(1Ø*K-8,5*J+5)
2280 A=1
2300 GOSUB7ØØØ
2500 NEXTI
2800 PRINT@66,"A    B    C    D    E    F    G    H";
2900 RETURN
3000 'ZIEHEN
3100 PRINT@1,"";:INPUT"STARTPOSITION";L$,M
3101 L=ASC(L$):L=L-64
3102 GOSUB28ØØ
3105 P=L+2:Q=M+1:GOSUB9ØØØ:IFI=ØGOTO325Ø
3110 GOTO31ØØ
3130 PRINT@4Ø,"UNERLAUBTER ZUG";
3140 FORI=1TO15ØØ:NEXTI
3145 GOSUB28ØØ
```

```
3150 PRINT@55,"     ";:PRINT@40,"";:INPUT"NAECHTER ZUG";L$,M
3151 L=ASC(L$)-64
3155 GOSUB2800
3158 FORI=0TO1:FORJ=6TO8:SET(I,J):NEXTJ,I
3160 IFL=0GOSUB2000:GOTO3350
3210 GOSUB9000:ONI+1GOTO3250,3130
3220 PRINT@40,"DA WAREN SIE SCHON";:GOTO3140
3250 REM ERASE
3255 IF S=2THEN3300
3260 S=ABS(S-1)
3270 GOSUB200
3290 IFS=1RESET(W-4,V+4):RESET(W-3,V+4)
3295 IFS<>1SET(W-4,V+4):SET(W-3,V+4)
3300 P=L:Q=M:S=1
3310 IF INT((P+Q)/2)=(P+Q)/2S=0
3320 O=O+1:I=O+64:J=Q:K=P:GOSUB7000
3330 I=P+8*(Q-1)
3340 A(I)=1:A(O+64)=I
3350 V=5*Q+1
3360 W=10*(P-1)+5
3370 GOSUB200
3390 J=0:PRINT@40,"":GOSUB2800
3395 PRINT@9,"";
3400 FORL=P-2TOP+2:FORM=Q-2TOQ+2:GOSUB9000:IFI=0J=J+1
3410 NEXTM:NEXTL
3450 PRINT@0,"SIE HABEN"J"ERLAUBTE ZUEGE";:IFJ=0THEN3450
4800 GOTO3150
5050 CLS:PRINT@520,CHR$(23)"DIE SPRINGER-TOUR":FORI=0TO1000:
NEXT:CLS
5060 PRINT"DIES IST EIN KLASSISCHES SCHACHPROBLEM. DIE AUFGA
BE BESTEHT
5070 PRINT"DARIN, EINEN EINZELNEN SPRINGER UEBER SAEMTLICHE
FELDER EINES
5080 PRINT"SCHACHBRETTS ZU BEWEGEN, OHNE EIN FELD ZWEIMAL AU
FZUSUCHEN.
5090 PRINT"MAN KANN AUF EINEM BELIEBIGEN FELD BEGINNEN.
6020 PRINT:PRINT"DER COMPUTER WIRD DIE ZUEGE VERFOLGEN UND K
EINEN UNERLAUBTEN
6030 PRINT"ZUG ZULASSEN. DAS SPIEL ENDET, WENN KEIN UNBESUCH
TES FELD MEHR
6040 PRINT"ERREICHBAR IST. dIE EINZELNEN FELDER WERDEN DURCH
6050 PRINT"BUCHSTABEN (A-H) UND ZAHLEN (1-8) DARGESTELLT. DI
E ZUEGE GIBT
```

```
6Ø6Ø PRINT"MAN EIN, INDEM MAN DEN BUCHSTABEN UND DIE ZAHL DE
S ZIELFELDES
6Ø7Ø PRINT"DURCH EIN KOMMA GETRENNT EINTIPPT UND ´eNTER´ DRU
ECKT.
6Ø8Ø PRINT:INPUT"WENN SIE BEGINNEN WOLLEN, ´ENTER´ DRUECKEN
-";A$
65ØØ RETURN
7ØØØ R=I:I=I-64
7Ø2Ø PRINT@178,I;"FELDER";
7Ø4Ø I=I/5
7Ø5Ø I=24Ø+15*(I-INT(I))+64*INT(I)
7Ø6Ø IFI<>INT(I)I=INT(I)+1
7Ø7Ø PRINT@I,CHR$(64+K)J;
73ØØ I=R
75ØØ RETURN
9ØØØ I=Ø:IF(L<1)+(L>8)+(M<1)+(M>8)I=1:GOTO95ØØ
91ØØ IFABS((P-L)*(Q-M))<>2I=1:GOTO95ØØ
92ØØ IFA(L-8+8*M)<>ØI=2
95ØØ RETURN
```

Stingray

Stingray
Mit diesem Programm wird ein Bildschirmspiel erzeugt, wie man es von einer bestimmten Art Spielautomaten kennt. Ein „feindliches Raumschiff" muß dadurch abgeschossen werden, daß man es in den Bereich einer „Strahlwaffe" bringt. Zur Steuerung des eigenen Raumschiffs dienen vier eng beieinander liegende Tasten der Tastatur: J, K. U und M. Am besten geht es, wenn man J und K mit dem Zeige- und Mittelfinger der rechten Hand bedient, U und M mit den entsprechenden Fingern der linken Hand. Das Raumschiff, das man abschießen soll, bewegt sich in zufälligen kleinen Sprüngen hin- und her (Zeilen 260 bis 306) und verschwindet jeweils nach einer Zeitdauer von ebenfalls zufälliger Länge, worauf ein neues erscheint. Erschwerend wirkt, daß die Steuerbewegungen auch nicht gleichmäßig ausgeführt werden, sondern in Schritten von zufälliger Größe, wie man in den Zeilen 310 bis 340 erkennt. Der Computer zählt getroffene und entkommene Raumschiffe und gibt nach insgesamt 20 dieser Vorgänge eine Beurteilung über den Schützen ab. Die Erläuterungen erscheinen am Anfang des Programmes, können aber auch zwischendurch mit einem Druck auf die Leertaste aufgerufen werden. Deshalb stehen sie in einem GOSUB-Teil des Programmes.

```
1 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
2 'ALL RIGHTS RESERVED  DO NOT COPY
3 'P.O. BOX 49104  ATLANTA GA  30359
4 CLS:GOSUB500
5 CLEAR
10 CLS
20 TX=RND(100)+9:TY=RND(30)+3
25 CLS:Q1=RND(60):N1=0
30 PRINT@0,NR" FEINDLICHE RAUMSCHIFFE ZERSTOERT
   "NW" ENTKOMMEN
```

```
40 IFNW+NR=>20GOTO1500
100 PRINT@534,"(-";:PRINT@544,"-)";
105 N=992
110 PRINT@980,"+";:PRINT@994,"+";
120 PRINT@917,"+";:PRINT@929,"+";
130 PRINT@854,"+";:PRINT@864,"+";
140 PRINT@791,"+";:PRINT@799,"+";
150 PRINT@728,"+";:PRINT@734,"+";
160 PRINT@665,"+";:PRINT@669,"+";
170 PRINT@602,"+";:PRINT@604,"+";
180 PRINT@539,"+";
190 PRINT@980," ";:PRINT@994," ";
200 PRINT@917," ";:PRINT@929," ";
210 PRINT@854," ";:PRINT@864," ";
220 PRINT@791," ";:PRINT@799," ";
225 PRINT@728," ";:PRINT@734," ";
230 PRINT@667," ";:PRINT@665," ";:PRINT@669," ";
240 PRINT@603," ";:PRINT@602," ";:PRINT@604," ";
250 PRINT@539," ";
260 S=RND(2):IFS=1THENX=RND(3)ELSEX=-1*RND(3)
270 IFS=1THENY=RND(2)ELSEY=-1*RND(2)
300 N1=N1+1:IFN1=>Q1THENNW=NW+1:GOTO10
305 RESET(TX,TY):RESET(TX-1,TY+1):RESET(TX+1,TY+1)
306 TX=TX+X:TY=TY+Y:X=0:Y=0
307 A$=INKEY$
310 IFA$="J"THENX=-1*RND(15)
320 IFA$="K"THENX=RND(15)
330 IFA$="U"THENY=-1*RND(5)
340 IFA$="M"THENY=RND(5)
345 IFA$=" "THEN GOSUB500
346 IFTX+X-1<1ORTX+X+1>126THENGOTO10
347 IFTY+Y<1ORTY+Y+1>46GOTO10
350 SET(TX+X,TY+Y):SET(TX-1+X,TY+1+Y):SET(TX+1+X,TY+1+Y)
355 IFA$<>""THENRESET(TX,TY):RESET(TX-1,TY+1):RESET(TX+1,TY+
1):TX=TX+X:TY=TY+Y:Y=0:X=0
360 IFTX<52ORTX>58THEN380
370 IFTY=23ORTY=24ORTY=25ORTY=26THEN400
380 GOTO100
400 FORXX=1TO10:PRINT@538,"****":GOSUB1005:PRINT@538,"    ":
GOSUB1005:NEXTXX:NR=NR+1
410 GOTO10
500 PRINT@88,"* STINGRAY *
510 PRINT"DU BIST DER KOMMANDANT DES TAKTISCHEN RAUMJAEGERS
```

```
´STINGRAY´.
520 PRINT"DIE STINGRAY IST MIT EINER STRAHLENKANONE BEWAFFNE
T, DIE AUF
530 PRINT"EINEN PUNKT 50 KM VOR DEM RAUMSCHIFF GERICHTET IST
. WENN ES DIR
540 PRINT"GELINGT, EIN FEINDLICHES RAUMSCHIFF IN DIESE POSIT
ION ZU BRIN-
550 PRINT"GEN, WIRD ES ZERSTOERT.
560 PRINT:PRINT"DU BIST AUF EINER KAMPFMISSION IN FEINDLICHE
M RAUMGEBIET.
570 PRINT"DEINE AUFGABE IST ES, SOVIELE GEGNERISCHE RAUMSCHI
FFE WIE
580 PRINT"MOEGLICH ZU ZERSTOEREN. WENN EINEM FEINDSCHIFF EIN
 RAUMSPRUNG
590 PRINT"GELINGT, VERSCHWINDET ES UND ENTKOMMT.
600 PRINT:PRINT"DEIN SCHIFF LAESST SICH WIE FOLGT STEUERN:":
PRINT" J = LINKS      K = RECHTS":PRINT" U = AUFWAERTS  M =
ABWAERTS   ERKLAERUNGEN AUFRUFEN: LEERTASTE
604 PRINT"ZUM SPIELSTART EINE BELIEBIGE TASTE DRUECKEN";
606 ED$=INKEY$:IF ED$=""GOTO606
607 CLS
610 RETURN
1000 PRINT@0,""
1005 FORX=1TO25:NEXT:RETURN
1020 NW=NW+1:GOTO10
1500 IFNR<16PRINT:PRINT"NICHT SCHLECHT, ABER "NW" SIND ENTKO
MMEN.
1510 IFNR>15PRINT:PRINT"DU BIST EIN GUTER SCHUETZE, DU HAST
"NR" ERWISCHT!
1520 FORX=1TO2000:NEXT:CLS:GOTO4
```

Zahlen Ordnen

Zahlen Ordnen
Dieses Programm soll eine Routine zum Ordnen von Zahlen demonstrieren, die man sicher in vielen Programmen gut gebrauchen kann. Sie besteht vor allem aus zwei ineinander geschachtelten Schleifen, die so oft durchlaufen werden müssen, wieviele Zahlen man der Größe nach ordnen will.

Die Routine arbeitet mit einer Variablen (V), die zunächst auf einen Wert gesetzt wird, der kleiner ist als die niedrigste der zu ordnenden Zahlen. (Der Deutlichkeit halber hier: -10^{38} – Zeile 60). Beim Durchlauf durch die innere Schleife (Zeilen 70 – 90) wird er dann bis auf den Wert der höchsten Zahl angehoben. Dann wird diese Zahl aus der weiteren Bearbeitung herausgenommen, indem man in einer Hilfsmatrix (M(15)) das entsprechende Element gleich 1 setzt (Zeile 100). In der Zeile 80 wird nämlich auch immer überprüft, ob das entsprechende M(x) noch nicht auf 1 gesetzt wurde, ehe die Angleichung des V erfolgt. Dies kann man sooft wiederholen, bis alle Zahlen geordnet sind. Wenn es weniger oft geschieht, werden eben nur die entsprechende Anzahl der höchsten Nummern herausgesucht.

Der Rest des Programmes, u. a. zufällige Wahl der zu ordnenden Zahlen (Zeile 30) und Ausgabe der ungeordneten und geordneten Zahlen auf dem Bildschirm (Zeile 110) ist nur zur Verdeutlichung des Prinzips gedacht und kann den Erfordernissen eines größeren Programmes, das mit der Routine arbeitet, natürlich angepaßt werden.

```
10 CLS
20 DIM X(15), M(15)
30 FOR I=1 TO 15: X(I)=RND(100): NEXT
40 PRINT "PLATZ", "UNGEORDNET", "GEORDNET
50 FOR L=1 TO 15
60 V=-1E+38
70 FOR I=1 TO 15
80 IF M(I)=0 AND X(I)>=V THEN V=X(I): O=I
90 NEXT I
100 M(O)=1
110 PRINT L, X(L), V;
120 IF L<15 PRINT
130 NEXT L
140 GOTO 140
```

Geräusche und Musik

Geräusche und Musik

Daß man mit dem TRS-80 auch ohne Hilfsmittel Geräusche und Musik machen kann, wurde bereits an anderer Stelle dieses Buches erwähnt. Um zu vernünftigen Resultaten zu kommen, ist dabei Maschinenprogrammierung erforderlich. Der Zeitaufwand für die Verarbeitung von BASIC-Programmen, die zur Ausführung ja zeilenweise in Maschinensprache umgesetzt werden müssen, ist zu groß, als daß so mehr als nur sehr tiefe Töne zustandekommen. Nun, hier sind zwei Programme, wie man sieht in BASIC, aber doch zum Teil Maschinenprogramme. Beiden gemeinsam ist der letzte Teil mit den Zeilen 50000 bis 50260. Hierbei handelt es sich um ein Unterprogramm zum Laden einer Subroutine in den Speicher, die anschließend in bekannter Weise mit der Funktion USR(x) von einem anderen Programm aus aufgerufen werden kann. Die Idee dabei ist es, einem Programm, das man mit Geräuschen oder Musik ausstatten will, dieses Unterprogramm anzuhängen (daher auch die hohen Zeilennummern). Am Anfang des Hauptprogramms muß es dann mit der Anweisung GOSUB 50000 aufgerufen werden und lädt ein Maschinenprogramm, das sich als Zahlenreihe in den DATA-Zeilen 50200 bis 50260 befindet in einen Speicherbereich, den man natürlich vor dem Überspielen des gesamten Programms vom Cassettengerät reserviert haben muß (mindestesn 125 Byte).

Die Speicherplätze 16561 und 16562 im vom Computer selbst beanspruchten Speicherbereich enthalten (als Zwei-Byte-Zahl) einen Wert, der um 2 kleiner ist als die erste Adresse des reservierten Speicherbereichs. In Zeile 50002 wird diese Zahl ermittelt und dient dann als Anfangspunkt für das Einschreiben des Maschinenprogramms. In Zeile 50004 wird geprüft, ob das Maschinenprogramm schon bei einem früheren Lauf des Hauptprogrammes geladen wurde. Ist das der Fall, kehrt der Computer mittels RETURN sofort aus dem Unterprogramm zurück, da das Maschinenprogramm ja nur beim erstenmal geladen werden muß. Schließlich wird auch noch die Sprungadresse des Ma-

schinenprogramms in die entsprechende Stelle für die Funktion USR(x) geladen (Zeilen 50006 und 50008). Die liegt um 7 höher als die Anfangsadresse des Maschinenprogramms; die ersten sieben Bytes enthalten nämlich Zahlen, mit denen man durch entsprechende POKE-Befehle die Art der Töne bestimmen kann, die erzeugt werden: Die ersten beiden Adressen enthalten das niederwertige und höherwertige Byte der Tonhöhe, die nächsten beiden die niederwertigen und höherwertigen Byte der Tondauer. Mit der fünften Zahl können bis zu 255 Stufen einer Tonreihe d. h. bis 256 Töne gewählt werden, und die letzten beiden Adressen vor der Sprungadresse (dem eigentlichen Beginn des Maschinenprogramms) bestimmen abwechselnd den Tonhöhenunterschied zwischen zwei Tönen einer Reihe, so daß man durch geschickte Wahl der Zahlenwerte Sirenen, Martinshörner u. ä. nachahmen kann.

Hier einige Hinweise zum besseren Verständnis der verwendeten Kontrollzahlen: Die Töne werden durch Programmschleifen erzeugt, deren Anzahl und Länge durch die Zahlen bestimmt werden. Dementsprechend hängen Höhe und Dauer eines Tones zusammen: Je höher, desto kürzer bei gleicher Längenangabe. Durch die Stufenangaben werden der jeweiligen Tonhöhenzahl die entsprechenden Werte hinzuaddiert oder von ihnen abgezogen, und zwar folgendermaßen: Die Werte 1 bis 127 werden zugezählt, so daß sich ein niedrigerer Ton ergibt. Bei den Zahlen 129 bis 255 wird das Komplement zu 256 abgezogen (also 127 –1), so daß sich ein entsprechend höherer Ton ergibt – am wenigsten bei 255, am meisten bei 129. Bei 128 passiert wie bei 0 garnichts. Am besten probiert man die Sache selbst ein wenig aus, allerdings sind einige Einschränkungen zu beachten, wenn man nicht unkontrollierbare Ergebnisse erhalten will:
Bei den Tonstufen sollte man vorsichtig sein, wenn man mit den Abstufungen über die Grenzen der vorhandenen Tonhöhen hinausgeht (also z. B. von der Tonhöhe 100, niedriges Byte, 120 als Tonstufe abziehen). Es kommen zwar auch dann noch Töne heraus, aber die liegen dann irgendwo auf der Tonskala, wo man sie nicht erwartet.

Bei Verwendung des höherwertigen Byte für die Tonhöhe funktioniert die Stufung nicht richtig – man bekommt z. B. keine glatt auf- oder absteigenden Tonreihen heraus.

Die Zahl 10 als höherwertiges Byte für die Tonhöhe stellt in etwa den tiefsten, noch brauchbaren Ton dar; darunter hört man mehr oder weniger nur noch einzelne Impulse.

Auch ist Vorsicht geboten, wenn man bei stufenweise aufsteigender Tonhöhe an die Grenze der Tonhöhe überhaupt gelangt (1 im niederwertigenTonhöhen-Byte). Dann verfällt der Computer unter Umständen für kürzere oder längere Zeit in langsame Knack-Geräusche und kann nur noch mit dem RESET-Knopf wieder gestoppt werden. Ein Beispiel: Bei der Anfangstonhöhe 100 darf man höchstens 99 Stufen mit der kleinsten Schrittweite aufwärts, nämlich 255 in einer der beiden Tonstufenadressen vorsehen. Bei größeren Tonschritten ist es allerdings anders.

Und noch eine wichtige Einzelheit: Am Anfang enthält die erste Kontrolladresse (niederwertiges Tonhöhen-Byte) 50 und Position Nr. 3 (niederwertiges Tonlängen-Byte) 15.

Alle anderen Zahlen sind Null. Das ergibt einen kurzen Piep-Ton, den man also erhält, wenn man die Subroutine durch die USR-Funktion aufruft, ohne die Kontrollzahlen irgendwie zu verändern. Diese Zahlenkombination wird nach jedem Maschinenprogramm-Aufruf wieder in die entsprechenden Speicherpositionen geladen, man muß also jede andere Zusammenstellung vor jedem USR erneut in die sieben Speicherplätze bringen. Aber das wäre in jedem Fall nötig, da die Zahlen im Zuge der Abarbeitung des Maschinenprogramms alle auf Null heruntergezählt werden.

Bleibt noch zu ergänzen, wie man die Töne hörbar machen kann: Einfach durch Anschluß des Cassettengerät-Ausgangs mit einem 5-poligen Diodenkabel an den Tonband- oder Plattenspielereingang eines Radios, Verstärkers oder etwas ähnlichem.

Geräusche

Geräusche
Zum Ausprobieren der Möglichkeiten der Geräusch-Subroutine dient das erste der beiden Programme. Darin kommt als erstes das GOSUB zum Unterprogramm. Dann werden die sieben Kontrollzahlen samt ihren Speicherplätzen auf dem Bildschirm gezeigt. Nacheinander kann man beliebige Zahlen (zwischen 0 und 255, natürlich) wählen, die in die entsprechenden Speicherplätze gebracht werden (Zeile 230). Gleichzeitig werden die gewählten Zahlen auch noch als Variable bereitgehalten (P(0) bis P(6)). Läßt man die Maschinen-Subroutine ablaufen (Zeile 220), werden die Kontrollzahlen anschließend wieder in ihre Speicher-Positionen gebracht, weil das Maschinenprogramm sie ja bei jedem Durchgang löscht. So kann man jede gewählte Kombination beliebig oft auslösen. Zeile 15 dient übrigens dazu, die Anfangsinhalte der Speicherstellen für die Kontrollzahlen auch richtig auf dem Bildschirm abzubilden.

Will man ein bestimmtes Programm mit Geräuschen ausstatten, kann man nun folgendermaßen verfahren: Zuerst lädt man dieses Probierprogramm. Dann findet man durch Versuche heraus, durch welche POKE-Anweisungen die gewünschten Effekte hervorgerufen werden und notiert sie sich. Wichtig ist dabei, daß man mit Hilfe der Variablen TM, die von dem jeweils reservierten Speicherbereich und damit auch von der Anfangsadresse des Maschinenprogrogramms abhängt, unabhängig von festen Speicheradressen für die Kontrollzahlen wird.

Schließlich kann man das geplante Programm schreiben und die Zeilen bis 240 des Probierprogramms löschen. Dabei muß das neue Programm nur an geeigneter Stelle wieder ein GOSUB 50000 enthalten. Mit passenden POKE-Anweisungen und dem USR(x) lassen sich die gewünschten Geräuscheffekte dann im Programm auslösen.

Das Unterprogramm zum Laden des Maschinenprogramms bildet dadurch einen Teil des neuen Programms. Beim ersten Lauf nach Anschalten des Computers wird es aktiviert und lädt das Maschinenprogramm in den vorgesehenen Speicherbereich. Das bleibt dort, bis der Computer wieder ausgeschaltet wird.

```
10 GOSUB 50000
15 P(0)=50: P(2)=15
20 CLS: PRINT"STARTADRESSE:   "TM+2"   (BEREITS GELADEN)
30 PRINT: PRINT"SPEICHERPLAETZE FUER KONTROLLZAHLEN:
   INHALT:
100 GOSUB 120: GOTO 200
120 PRINT@192,"HOEHE (NIEDRIGES BYTE): "TM+2,,PEEK(TM+2)"
     "
130 PRINT"HOEHE (HOHES BYTE):      "TM+3,,PEEK(TM+3)
140 PRINT"DAUER (NIEDRIGES BYTE): "TM+4,,PEEK(TM+4)
150 PRINT"DAUER (HOHES BYTE):     "TM+5,,PEEK(TM+5)
160 PRINT"STUFENZAHL:             "TM+6,,PEEK(TM+6)
170 PRINT"TONSTUFE 1:             "TM+7,,PEEK(TM+7)
180 PRINT"TONSTUFE 2:             "TM+8,,PEEK(TM+8)
190 RETURN
200 PRINT: PRINT"SUBROUTINE AENDERN: 1,   AKTIVIEREN: 2
210 A$=INKEY$: IF A$="" THEN 210
220 IF A$="2" Z=USR(0): FOR I=0 TO 6: POKE TM+2+I,P(I): NEXT
: GOTO 210
230 IF A$="1" THEN FOR I=0 TO 6: PRINT@ 245+64*I,;:INPUT P(I
):   POKE TM+2+I,P(I): GOSUB 120: NEXT
240 GOTO 210
50000 'GERAEUSCH-SUBROUTINE - LADEPROGRAMM
50002 TM=PEEK(16561)+256*PEEK(16562)
50004 IF PEEK(TM+9)=219 THEN RETURN
50010 CLS: PRINT"'MEMORY SIZE' MUSS MINDESTENS 125 BYTE NIED
RIGER ALS DAS
50015 PRINT"MAXIMUM DES COMPUTERS - D. H. HOECHSTENS 32642 B
EI 16 K
50016 PRINT"BZW. 20324 BEI 4 K SPEICHERGROESSE - EINGESTELLT
 SEIN!
50020 PRINT"IST DAS NICHT GESCHEHEN, COMPUTER AUSSCHALTEN UN
D NEU BEGINNEN!
50027 INPUT"ALLES KLAR? DANN 'ENTER DRUECKEN' -";
50030 PRINT: PRINT"EINEN MOMENT...
50040 N=16526: ML=TM+9: GOSUB 50130
```

```
50050 READ A$: IF A$<>"ML$" THEN 50050
50060 FOR N=TM+2 TO TM+200
50070 READ ML: IF ML>255 THEN 50110
50080 IF ML<0 THEN GOSUB 50120: GOTO 50100
50090 POKE N,ML
50100 NEXT
50110 RESTORE: RETURN
50120 ML=TM-ML+1
50130 Z1=INT(ML/256): Z2=ML-Z1*256
50140 POKE N,Z2: N=N+1: POKE N,Z1: RETURN
50200 DATA ML$,50,0,15,0,0,0,0,219,255,230,64,238,64,15,15,1
5,   246,1,95,243,58,-6,87,58,-7,254,0,194,-37,122,50,-7
50220 DATA 42,-1,34,-53,42,-3,34,-50,33,15,0,1,50,0,11,120,1
77,  194,-55,123,238,3,211,255,95,43,124,181,194,-52,58
50240 DATA -5,254,0,202,-104,61,50,-5,58,-53,130,50,-53,58,-
7,   103,122,50,-7,84,195,-49,50,-7,50,-4,50,-2,62,15
50260 DATA 50,-3,62,50,50,-1,251,201,300
```

Musik

Musik

Daß sich recht reizvolle Effekte erzielen lassen, ohne daß man dazu alle der vorhandenen Beeinflussungsmöglichkeiten des Musik-Maschinenprogramms nutzt, zeigt dieses zweite Programm, das mit dem Unterprogramm arbeitet. Hier werden nur Tonhöhe und -länge verwendet, und damit kann man ganze Melodien komponieren und abspielen.

In Zeile 20 ist die Zahl der Noten festgelegt: Zwei Matrizen mit je zweimal 10 Elementen können die jeweils zwei Tonhöhen- und Tonlängen-Bytewerte von 100 Noten bzw. die Längen von Pausen aufnehmen (nur hundert Elemente werden verwendet). Nacheinander werden diese Werte in die entsprechenden Speicherplätze gepoked und die USR(x)-Funktion aufgerufen, um die eingegebene Melodie abzuspielen (Zeile 90). Für Pausen wird einfach eine passend dimensionierte Warteschleife aktiviert (Zeile 85). Beim „Komponieren" nimmt der Computer die bekannten Tonbezeichnungen entgegen (C, Cis, Des, D usw.) und wählt die entsprechenden Tonhöhen aus einer Liste, die in den DATA-Zeilen 3000 bis 3008 vorgegeben wird. Übrigens enthält jede der DATA-Zeilen genau eine Oktave von C bis H. Diese Zahlen lassen sich auch hernehmen, wenn man irgendein anderes Programm im Sinn hat, das eine Melodie enthalten soll.

Da Tonhöhen und -längen voneinander abhängen, können eingegebene Längen nicht einfach übernommen werden, sondern müssen anhand der jeweils angegebenen Höhe zuerst umgerechnet werden. Dies geschieht unter Verwendung der Referenzzahl RZ, die in Zeile 10 definiert wird, in Zeile 400 und zwar so, daß das Produkt aus der Multiplikation von Tonhöhen und -längen für eine bestimmte Tonlänge immer eine Konstante ergibt. In Zeile 330 und in dem zweiten und dritten Teil von Zeile 400 finden sich übrigens die bekannten Umrechnungen von Größen in ihre Darstellungen durch zwei Byte.

Vielleicht wird sich mancher wundern, warum die Zeile 30 mit einem RESTORE abschließt, obwohl das Übernehmen von Zahlen aus DATA-Zeilen in dem anschließenden, durch GOSUB 50000 angesprochenen Unterprogramm zumindest bei erstmaligem Programmlauf weitergeht. Nun, es ist grundsätzlich empfehlenswert, verschiedene Teile eines Programms, die jeder für sich DATA-Zeilen und READ-Befehle haben, sinngemäß so auseinanderzuhalten, daß jederzeit neue Teile hinzugefügt oder herausgenommen werden können, ohne daß man sich Sorgen machen muß, ob die einzelnen READ-Anweisungen auch jeweils die richtigen DATA-Zeilen erreichen. Dazu muß am Ende jedes READ-Vorgangs ein RESTORE stehen, und am Anfang sollte eine „Sicherung" eingebaut sein, die den Zugriff von READ-Befehlen auf falsche Daten verhindert. Letzteres geschieht beim Unterprogramm in Zeile 50050, in Verbindung mit dem ersten Begriff in den zugehörigen Data-Zeilen, dem String ML$. Etwaige DATA-Zeilen im Gesamtprogramm werden so lange abgesucht, bis das ML$ gefunden ist; erst dann beginnt das Übertragen des Maschinenprogramms in seinen Speicherbereich.

Eine erwähnenswerte Besonderheit haben übrigens die beiden hier abgedruckten Programme: In bedingten Befehlen steht bei ihnen vor Sprungbefehlen weder THEN noch GOTO, sondern einfach ein Komma – im Musikprogramm zum Beispiel unter anderem in den Zeilen 70, 75 und 80. Wo dies ohne Widersprüche möglich ist, kann sogar das Komma entfallen. Daß dies funktioniert, beweisen die Programme. Es handelt sich dabei um eine angenehme Arbeitserleichterung beim Programmieren, die noch garnicht so lange bekannt ist!

```
10 DEFINT T: RZ=4096
20 DIM TP(100,1). TL(100,1), TZ(5,11)
25 TL(1,0)=500
30 FOR K=1 TO 5: FOR J=0 TO 11: READ TZ(K,J): NEXT J,K: REST
ORE
40 GOSUB 50000: ST=TM÷2
50 CLS: PRINT"MUSIK ! ! ! ! !
60 PRINT: PRINT"1 - KOMPONIEREN     2 - AENDERN     3 - SPIE
LEN
70 A$=INKEY$: IF A$="", 70
75 IF A$<>"3", 95
80 FOR K=1 TO 100: IF TL(K,0)>255, 70
82 PRINT@58, K
85 IF TP(K,0)=0 AND TP(K,1)=0 FOR J=1 TO TL(K,0)*20: NEXT J,
K
```

```
9Ø POKE ST,TP(K,Ø): POKE ST+1,TP(K,1): POKE ST+2,TL(K,Ø): PO
KE ST+3,TL(K,1): M=USR(M): NEXT K: GOTO 7Ø
95 IF A$="2" CLS: INPUT"AENDERUNGEN - BEI WELCHER POSITION B
EGINNEN";KØ: GOTO 165
1ØØ IF A$<>"1", 7Ø
11Ø CLS: PRINT"ALSO LOS, DU MEISTERKOMPONIST!
12Ø PRINT: PRINT"DIE NOTEN WERDEN NACHEINANDER EINGEGEBEN -
13Ø PRINT"IN DER FORM: NOTE 'ENTER' OKTAVE (1-5), LAENGE (1-
5Ø) 'ENTER'.
135 PRINT"ES GIBT NOCH EINEN TON VON DER 6. OKTAVE: DAS HOHE
 C.
14Ø PRINT"Z. b.: G 'ENTER' 2,8 'ENTER' CIS 'ENTER' 3,12 'ENT
ER' USW.
15Ø PRINT"PAUSEN LEGT MAN MIT 'P' UND DER DAUER (1-5Ø) FEST.
152 PRINT"FUER DAS ENDE DER MELODIE X EINGEBEN.
153 PRINT"EIN 'Q' BRICHT KOMPOSITIONS- UND AENDERUNGSROUTINE
N AB.
155 PRINT: INPUT"ALLES KLAR? DANN 'ENTER' DRUECKEN";:CLS
16Ø PRINT"UND NUN FRISCH ANS WERK!
163 KØ=1
165 FOR K=KØ TO 1ØØ: I$="": I1=Ø: I2=Ø
19Ø PRINT@256, CHR$(31) "NOTE/PAUSE NR."K;: INPUT I$
191 IF I$="" AND A$="2" NEXT: GOTO 5Ø
192 IF I$="Q", 5Ø
193 IF I$="X" TL(K,Ø)=5ØØ: GOTO 5Ø
195 IF I$="P" INPUT"LAENGE"; I2: TP(K,Ø)=Ø: TP(K,1)=Ø: TL(K,
Ø)=  I2: NEXT ELSE INPUT"OKTAVE, LAENGE"; I1,I2
21Ø IF I1=6 PG=16: GOTO 33Ø
24Ø Y=12
25Ø IF I$="C" Y=Ø
255 IF I$="CIS" OR I$="DES" Y=1
26Ø IF I$="D" Y=2
265 IF I$="DIS" OR I$="ES" Y=3
27Ø IF I$="E" Y=4
275 IF I$="F" Y=5
28Ø IF I$="FIS" OR I$="GES" Y=6
285 IF I$="G" Y=7
29Ø IF I$="GIS" OR I$="AS" Y=8
295 IF I$="A" Y=9
3ØØ IF I$="AIS" OR I$="B" Y=1Ø
3Ø5 IF I$="H" Y=11
31Ø IF Y=12, 19Ø
32Ø PG=TZ(I1,Y)
```

```
330 PH=INT(PG/256): PL=PG-PH*256
340 TP(K,0)=PL: TP(K,1)=PH
400 RL=I2*RZ/PG: LH=INT(RL/256): LL=RL-256*LH
410 TL(K,0)=LL: TL(K,1)=LH
600 NEXT K: GOTO 70
999 END
3000 DATA 560,524,500,476,442,420,396,376,352,336,316,298
3002 DATA 280,262,250,238,221,210,198,188,176,168,158,149
3004 DATA 140,131,125,119,111,105,99,94,88,83,79,74
3006 DATA 70,66,62,59,55,52,49,46,44,41,39,36
3008 DATA 34,32,30,28,26,25,24,22,21,19,18,17
50000 'GERAEUSCH-SUBROUTINE - LADEPROGRAMM
50002 TM=PEEK(16561)+256*PEEK(16562)
50004 IF PEEK(TM+9)=219 THEN RETURN
50010 CLS: PRINT"'MEMORY SIZE' MUSS MINDESTENS 125 BYTE NIED
RIGER ALS DAS
50015 PRINT"MAXIMUM DES COMPUTERS - D. H. HOECHSTENS 32642 B
EI 16 K
50016 PRINT"BZW. 20324 BEI 4 K SPEICHERGROESSE - EINGESTELLT
 SEIN!
50020 PRINT"IST DAS NICHT GESCHEHEN, COMPUTER AUSSCHALTEN UN
D NEU BEGINNEN!
50027 INPUT"ALLES KLAR? DANN 'ENTER DRUECKEN' -";
50030 PRINT: PRINT"EINEN MOMENT...
50040 N=16526: ML=TM+9: GOSUB 50130
50050 READ A$: IF A$<>"ML$" THEN 50050
50060 FOR N=TM+2 TO TM+200
50070 READ ML: IF ML>255 THEN 50110
50080 IF ML<0 THEN GOSUB 50120: GOTO 50100
50090 POKE N,ML
50100 NEXT
50110 RESTORE: RETURN
50120 ML=TM-ML+1
50130 Z1=INT(ML/256): Z2=ML-Z1*256
50140 POKE N,Z2: N=N+1: POKE N,Z1: RETURN
50200 DATA ML$,50,0.15,0,0,0,0,219,255,230,64,238,64.15,15,1
5,   246,1.95,243.58,-6,87,58,-7,254,0,194,-37,122,50,-7
50220 DATA 42,-1,34,-53,42,-3,34,-50,33,15,0,1,50,0,11,120,1
77,  194,-55,123,238,3,211,255,95,43,124,181,194,-52,58
50240 DATA -5,254,0,202,-104,61,50,-5,58,-53,130,50,-53,58,-
7,   103,122,50,-7,84,195,-49,50,-7,50,-4,50,-2,62,15
50260 DATA 50,-3,62,50,50.-1,251,201,300
```

Byte-Zerlegung von Zahlen

Byte-Zerlegung von Zahlen
Nicht selten muß man Speicheradressen in zwei Byte zerlegen, um sie direkt in entsprechende Speicherpositionen des Computers zu bringen. Ein Beispiel dafür bildet die Funktion USR(x), die eine Maschinenprogramm-Subroutine aufruft und dazu die Angabe einer Sprungadresse erfordert, die man mit POKE-Befehlen in die Speicherstellen 16526 und 16527 befördern muß. Um zeitraubendes Ausrechnen der beiden Byte-Werte zu ersparen, ist dieses kleine Programm gedacht. Bei Eingabe von entsprechenden Dezimalzahlen ermittelt es die entsprechende Byte-Darstellung. Die dazu erforderliche Kalkulation findet in Zeile 110 statt. Das restliche Programm ist größtenteils nicht unbedingt erforderliche, aber recht praktische „Ausschmückung". Zunächst wird die umzurechnende Zahl in Zeile 30 als ganze Zahl definiert. Das hilft Fehler vermeiden, da dadurch nur Zahlen bis 32767 – der höchsten Adresse des TRS-80 mit 16 K Speicher – verarbeitet werden. Dann sorgt die Formationsdefinition in Zeile 40 in Verbindung zu den entsprechenden USING-Angaben für die korrekte Ausrichtung der auf dem Bildschirm erscheinenden Zahlen. Schließlich wird mit der ON ERROR GOTO-Routine und dem ?REDO der Zeile 200 die Eingabe-Fehlermeldung des Computer-Betriebsystems nachgeahmt. Mit dem CHR$ (27) CHR$(30) erreicht man, daß nach Eingabe der zu zerlegenden Zahl sie zur Kontrolle auf der gleichen Zeile des Bildschirms wieder sichtbar wird.

```
10 CLS
20 ON ERROR GOTO 200
30 DEFINT X
40 F$="#####"
100 PRINT: INPUT "DEZIMALZAHL"; X
110 H=INT(X/256): L=X-H*256
130 PRINT CHR$(27)CHR$(30) "DEZIMALZAHL: ",, USING F$; X
140 PRINT "HOEHERWERTIGES BYTE: ", USING F$; H
150 PRINT "NIEDERWERTIGES BYTE: ", USING F$; L
160 GOTO 100
200 PRINT "?REDO
210 RESUME
```

Geräuschprogramm ohne Maschinenprogramm-Speicherbereich

Geräuschprogramm ohne Maschinenprogramm-Speicherbereich
Dieses Programm ist eine Variante des Probierprogramms, mit dem Unterschied, daß man keinen Speicherplatz für ein Maschinenprogramm reservieren muß. Das wird dadurch erreicht, daß eine Zeichenkette zur Aufnahme des Maschinen-Unterprogramms vorgesehen wird. Diese Zeichenkette wird in Zeile 50001 definiert — sie ist 125 Stellen lang und kann damit gerade das Maschinenprogramm aufnehmen. In diesen Platz wird in bekannter Weise mit einer Programmschleife, in der jeweils READ- und POKE-Anweisungen aufeinander folgen, das Maschinenprogramm eingetragen. In den Speicherplatz für die Sprungadresse der USR-Funktion (16526 und 16527) wird in Zeile 50008 die Anfangsadresse des String geladen, die in Zeile 50002 gewonnen wurde. Dadurch wird das Maschinenprogramm aktiviert, wenn das Programm USR(x) verarbeitet (Zeile 220).

Als Nachteil gegenüber der Variante mit in eigenem Speicherbereich untergebrachten Maschinenprogramm läßt sich dieses Programm weder korrekt listen noch wieder starten, wenn es erst einmal gelaufen ist; der Inhalt des String T$ bringt die Computer-Funktion durcheinander. Als Abhilfe kann man nach einem BREAK mit folgender Befehlszeile für Programmlistungen sowie neue Starts wieder geordnete Verhältnisse schaffen: FORI=17919 TO 18043: POKE I, 50: NEXT.

Die Anfangsadresse gilt natürlich nur für den Fall, daß das Programm exakt wie in der Vorlage eingegeben wurde. Ansonsten wird die Anfangsadresse ja auf dem Bildschirm sichtbar gemacht. Die Zahl 50 in dem POKE-Befehl ist eine zufällige Wahl; selbstverständlich ist auch eine beliebige andere Zahl geeignet, die im Bereich 1–255 liegt und irgendein Zeichen repräsentiert.

Die bei Zeile 50000 beginnende Subroutine zum Laden des Maschinenprogramms in eine freie Zeichenkette legt wieder den Gedanken nahe, das Unterprogramm wiederum in anderen BASIC-Programmen einzusetzen. Dafür ist es auch gedacht. Es muß am Anfang des übergeordneten Programmes nur einmal aufgerufen werden (mit einem GOSUB 50000), damit das Maschinenprogramm in den DATA-Zeilen in die Zeichenkette T$ in Zeile 50001 übertragen wird.

Dann kann ,wie bereits ausgeführt, die Art des Tones bzw. der Töne durch entsprechende POKE-Anweisungen gewählt werden, wobei die Variable ST als Bezugspunkt fungiert. Sie wird in Zeile 50002 mit Hilfe der VARPTR-Funktion aus der Anfangsadresse von T$ gewonnen und dient auch zum Laden der Sprungadresse der USR-Funktion (Zeile 50008). Daher kann die Subroutine in gewohnter Weise durch X= USR(X) aufgerufen werden.

```
5 CLS: PRINT "EINEN MOMENT, BITTE...
10 GOSUB 50000
15 P(0)=50: P(2)=15
20 CLS: PRINT"STARTADRESSE:   "ST"   (BEREITS GELADEN)
30 PRINT: PRINT"SPEICHERPLAETZE FUER KONTROLLZAHLEN:
   INHALT:
100 GOSUB 120: GOTO 200
120 PRINT@192,"HOEHE (NIEDRIGES BYTE): "ST,,PEEK(ST)"
  "
130 PRINT"HOEHE (HOHES BYTE):      "ST+1,,PEEK(ST+1)
140 PRINT"DAUER (NIEDRIGES BYTE): "ST+2,,PEEK(ST+2)
150 PRINT"DAUER (HOHES BYTE):      "ST+3,,PEEK(ST+3)
160 PRINT"STUFENZAHL:              "ST+4,,PEEK(ST+4)
170 PRINT"TONSTUFE 1:              "ST+5,,PEEK(ST+5)
180 PRINT"TONSTUFE 2:              "ST+6,,PEEK(ST+6)
190 RETURN
200 PRINT: PRINT"SUBROUTINE AENDERN: 1.   AKTIVIEREN: 2
210 A$=INKEY$: IF A$="" THEN 210
220 IF A$="2" Z=USR(0): FOR I=0 TO 6: POKE ST+I,P(I): NEXT:
GOTO 210
230 IF A$="1" THEN FOR I=0 TO 6: PRINT@ 245+64*I,;:INPUT P(I
):   POKE ST+I,P(I): GOSUB 120: NEXT
240 GOTO 210
50000 'GERAEUSCH-SUBROUTINE - LADEPROGRAMM
50001 T$="......................................................
....................................................................
................"
50002 ST=PEEK(VARPTR(T$)+1)+256*PEEK(VARPTR(T$)+2)
50004 BB=ST+7: BH=INT(BB/256): BL=BB-BH*256
50008 POKE 16526,BL: POKE 16527,BH
50050 READ A$: IF A$<>"ML$" THEN 50050
50060 FOR N=ST TO ST+200
50070 READ ML: IF ML>255 THEN 50110
50080 IF ML<0 THEN 50120
50090 POKE N,ML
50100 NEXT
50110 RESTORE: RETURN
50120 ML=ST-ML-1
50130 Z1=INT(ML/256): Z2=ML-Z1*256
50140 POKE N.Z2: N=N+1: POKE N.Z1: GOTO 50100
50200 DATA ML$,50,0,15,0,0,0,0,219.255,230.64.238,64,15.15.1
5.   246.1,95,243,58,-6.87,58,-7.254,0,194.-37,122.50,-7
50220 DATA 42,-1,34,-53.42.-3,34,-50,33,15.0,1.50,0,11,120,1
```

```
77,  194,-55,123,238,3,211,255,95,43,124,181,194,-52,58
50240 DATA -5,254,0,202,-104,61,50,-5,58,-53,130,50,-53,58,-
7,   103,122,50,-7,84,195,-49,50,-7,50,-4,50,-2,62,15
50260 DATA 50,-3,62,50,50,-1,251,201,300
```

Abgezinste Geldmengen

Abgezinste Geldmengen

Dieses Programm ermittelt, wie man auch an dem Anfangstext erkennen kann, den auf die Gegenwart abgezinsten Betrag einer Geldmenge in der Zukunft. Das heißt, wenn man weiß, wie hoch die Zinsen sind, die man bekommen kann, wenn man eine Geldsumme anlegt, kann dieses Programm ermitteln, wieviel Geld man heute anlegen muß, um zu einem bestimmten Zeitpunkt in der Zukunft eine gewünschte Geldmenge in Händen zu haben.

Zu dem Programm selbst ist nicht viel zu sagen: Die Angabe dererrechneten Lösung erfolgt mit einer Format-Anweisung (Zeile 1160), so daß in korrekter Weise Mark und Pfennig herauskommen. Auch ist eine Fehlerbehandlungsroutine vorhanden, die u. a. bei Eingabefehlern eingreifen soll. Die für einen Rechengang eingegebenen Anfangswerte werden bei Ausgabe der Lösung noch einmal sichtbar gemacht. So hat man alles auf dem Bildschirm beieinander und kann zum Beispiel für einen neuen Durchlauf im Vergleich damit andere Anfangswerte wählen.

```
1 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
2 'ALL RIGHTS RESERVED.  DO NOT COPY.
3 'P.O. BOX 49104 ATLANTA GA 30359
100 '
1000 F=0:R=0:M=0:N=0
```

```
1 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
2 'ALL RIGHTS RESERVED.  DO NOT COPY.
3 'P.O. BOX 49104 ATLANTA GA 30359
100 '
1000 F=0:R=0:M=0:N=0
1010 CLS:PRINT:PRINT"DIESES PROGRAMM BERECHNET DEN AUF DIE g
EGENWART ABGEZINSTEN
1020 PRINT"WERT EINER KUENFTIGEN gELDSUMME, UND ZWAR FUER EI
NEN BESTIMMTEN
1030 PRINT"ZINSFUSS.
1040 CLEAR:PRINT:INPUT"DER KUENFTIGE BETRAG, IN d-MARK";F
1050 INPUT"DER JAEHRLICHE zINSSATZ, IN %";R
1060 INPUT"ANZAHL DER ZINSPERIODEN PRO jAHR (GEWOEHNLICH 12)
";M
1070 INPUT"ANZAHL DER ZINSPERIODEN DES BERECHNUNGSZEITRAUMS"
;N
1075 ONERRORGOTO13000
1080 I=R/M:I=I/100
1090 T=1+I:A=T
1095 IFN=1GOTO1110
1100 FORX=1TO(N-1):S=A*T:A=S:NEXTX
1120 P=F/A:CLS:PRINT:PRINT:PRINT
1130 PRINT"EIN BETRAG VON"F"DM, ABGEZINST UEBER"N"ZINSPERIOD
EN,
1140 PRINT"BEI"M"JAEHRLICHEN ZINSPERIODEN UND EINEM JAHRES-Z
INSSATZ
1150 PRINT"VON"R"%, ENTSPRICHT DEM ANFANGSWERT VON
1160 PRINT:PRINTTAB(30)USING"#######.## DM";P
1170 PRINT:INPUT"FUER EINE NEUE BERECHNUNG 'ENTER' DRUECKEN
-";A$
1180 CLS:GOTO1040
1190 END
13000 CLS:IFERR/2+1=11THENPRINT"ES IST EIN FEHLER AUFGRUND E
INES VERSUCHS AUFGETRETEN,"ELSE13050
13010 PRINT"DURCH NULL ZU TEILEN. NORMALERWEISE LIEGT DAS AN
 EINER
13020 PRINT"FEHLERHAFTEN dATENEINGABE.
13030 PRINT"WIR WOLLEN NOCH EINMAL BEGINNEN. wENN sIE SOWEIT
 SIND,
13040 INPUT"BITTE 'ENTER' DRUECKEN - ";A$:GOTO100
13050 PRINT"EIN FEHLER IST PASSIERT. FANGEN WIR NOCH EINMAL
VON VORN AN.
13060 INPUT"BITTE 'ENTER' DRUECKEN - ";A$:GOTO100
```

Rückzahlung eines Darlehens in gleichen Raten

Rückzahlung eines Darlehens in gleichen Raten
Dieses zweite Programm für persönliche Finanzen gibt für einen in gleichen Monatsraten rückzahlbaren Kredit in Abhängigkeit vom Jahreszins die Höhe der einzelnen Raten an und teilt jede einzelne Rate in Zins- und Tilgungsanteil auf. So kann man sich vom Computer ausrechnen lassen, wie hoch die monatliche Belastung sein wird, wenn man eine bestimmte Geldsumme als Darlehen aufnimmt und innerhalb einer bestimmten Zeitspanne abzahlen will. Die Aufteilung der Raten in Zins und Tilgung kann wichtig sein, wenn man zum Beispiel – was meist möglich ist – die Zinsen von der Steuer absetzen will. Allerdings hat dies für normale Konsumkredite keine Bedeutung, da für diese Darlehensart feste monatliche Nominalzinswerte angegeben werden, die dann beim Finanzamt geltend gemacht werden können.

Das Programm selbst weist wie das vorherige formatierte Ausgabe der Lösungen auf. In diesem Fall wird die dazu dienende Anweisung in Zeile 4000 als String definiert, da sie mehrmals gebraucht wird (Zeilen 4140 und 41450).

```
1 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
2 'ALL RIGHTS RESERVED.  DO NOT COPY.
3 'P.O. BOX 49104  ATLANTA GA  30359
100 '
```

```
4ØØØ CLEAR:          :C=Ø:P=Ø:L=Ø:R=Ø:M=Ø:F$="#######.##":CLS
4ØØ2 PRINT"DIESES PROGRAMM ERMITTELT DIE RATEN EINES IN GLEI
CHEN
4ØØ4 PRINT"MONATSRATEN RUECKZAHLBAREN DARLEHENS UND GIBT FUE
R
4ØØ6 PRINT"JEDE RATE DEN JEWEILIGEN ZINS- UND TILGUNGSANTEIL
 SOWIE
4ØØ8 PRINT"DIE VERBLEIBENDE DARLEHENSSUMME AN. WAHLWEISE KAN
N AUCH
4ØØ9 PRINT"EINE TILGUNGSRATE EINGEGEBEN WERDEN.
4Ø1Ø PRINT:INPUT"DARLEHENSSUMME";P
4Ø2Ø INPUT"ANZAHL DER MONATSRATEN";L
4Ø3Ø INPUT"JAHRES-ZINSSATZ (%)";R
4Ø35 INPUT"TILGUNGSRATE (FALLS GEWUENSCHT, SONST NUR 'ENTER'
)";M
4Ø4Ø I=R/12ØØ
4Ø5Ø T=1-1/(1+I)[L:K=P
4Ø7Ø IFM<>ØTHENGOTO4Ø9Ø
4Ø75 ONERRORGOTO13ØØØ
4Ø8Ø M=P*I/T
4Ø9Ø GOSUB42ØØ
41ØØ FORZ=1TOL
411Ø IFC<12GOTO4117
4113 PRINT"DARLEHENSJAHR:"INT(Z-1)/12;
4114 PRINT" - "K"DM FUER"L"MONATE ZU"R"%
4115 INPUT"WEITER: 'ENTER' - NEUE BERECHNUNG: 1 + 'ENTER'";C
H:IFCH=1THEN1ØØ
4116 C=Ø:GOSUB42ØØ
4117 A=P*I
413Ø B=M-A:P=P-B
414Ø PRINTZTAB(8)USINGF$;P;:PRINTTAB(2Ø)USINGF$;M;
415Ø PRINTTAB(3Ø)USINGF$;B;:PRINTTAB(4Ø)USINGF$;A
416Ø C=C+1:NEXTZ
417Ø PRINT:INPUT"FUER EINEN NEUEN RECHENGANG 'ENTER' DRUECKE
N -";A$:GOTO1ØØ
418Ø END
42ØØ CLS
425Ø PRINT"RATE      RESTLICHE    MONATL.   TILGUNGS-  ZINS-
```

```
4260 PRINT"NUMMER    DARL/SUMME   ZAHLUNG   ANTEIL    ANTEI
L
4270 RETURN
13000 CLS:IFERR/2+1=11THENPRINT"INFOLGE EINES VERSUCHS, DURC
H NULL ZU TEILEN, IST EIN FEHLER"ELSE13050
13010 PRINT"AUFGETRETEN. DAS DEUTET AUF EINE FEHLERHAFTE DAT
ENEINGABE.
13020 PRINT"WIR WOLLEN NOCH EINMAL VON VORN BEGINNEN.
13030 INPUT"WENN SIE SOWEIT SIND, BITTE /ENTER/ DRUECKEN - "
;A$: GOTO100
13050 PRINT"ES IST EIN FEHLER AUFGETRETEN. WIR WOLLEN NOCH E
INMAL
13060 INPUT"ANFANGEN. BITTE /ENTER/ DRUECKEN";A$: GOTO 100
```

Effektivzins



Effektivzins
Dies ist ein sehr praktisches Programm, das aus der Höhe eines Darlehens sowie seinen Rückzahlungsbedingungen (gleiche Raten vorausgesetzt) den tatsächlich geforderten Zins ermittelt, der ja bekanntlich erheblich von dem Normalzinssatz abweichen kann. Banken sind zwar heutzutage verpflichtet, den Effektivzinssatz anzugeben, zu dem sie Geld an ihre Kunden verleihen. Bei anderen Stellen kann man das aber nicht immer voraussetzen.

Als Besonderheit gewährt das Programm die Möglichkeit, nach einem Rechengang einzelne Angaben zu verändern und damit eine neue Rechnung durchzuführen. Dazu dienen die Zeilen 8150 bis 8260. Ansonsten ähnelt es den vorherigen. Die Rechnung wird wieder nach den bekannten Zinsformeln durchgeführt.

```
6000 CLEAR:F=0:CLS:A$="###.## %"
6010 ONERRORGOTO13000
6012 PRINT"DIESES PROGRAMM ERRECHNET AUS DARLEHENSSUMME SOWI
E ANZAHL,
6014 PRINT"HOEHE UND ZEITLICHEM ABSTAND GLEICHER RUECKZAHLUN
GEN
6016 PRINT"DEN JAEHRLICHEN EFFEKTIVZINS EINES DARLEHENS. NEU
E
6018 PRINT"RECHENGAENGE LASSEN SICH NACH AENDERUNG EINZELNER
6019 PRINT"ANGABEN DURCHFUEHREN.
6020 PRINT:INPUT"DARLEHENSSUMME (D-MARK)";PV
6025 IFF=1GOTO6070
6030 INPUT"ANZAHL DER JAEHRLICHEN ZAHLUNGEN";NY
6035 IFF=3GOTO6070
6040 INPUT"ANZAHL DER INSGESAMT ZU LEISTENDEN ZAHLUNGEN";N
6045 IFF=2GOTO6070
6050 INPUT"HOEHE DER EINZELNEN RATEN (D-MARK)";P
6055 IFF=4GOTO6070
6060 CLS:PRINT"EINEN MOMENT...
6070 I=.009
6080 I1=P/PV*((1+I)[N-1)/(1+I)[N
6090 IFABS(I-I1)<.000001THEN6140
6100 I=I1
6110 GOTO6080
6140 I=I1*NY*100
6150 CLS:PRINT
6160 PRINT"EIN DARLEHEN UEBER"PV"DM,
6170 PRINT"DAS IN"N"RATEN VON JE"P"DM
6180 PRINT"BEI"NY"JAEHRLICHEN ZAHLUNGEN RUECKZAHLBAR IST,
6190 PRINT"HAT EINEN EFFEKTIVEN JAHRESZINSSATZ VON
6200 PRINTTAB(28)USINGA$;I
6210 F=0:PRINT
6220 PRINT"WAS WOLLEN SIE FUER EINEN NEUEN RECHENGANG AENDER
N?
6230 PRINT"   DARLEHENSSUMME","1
6240 PRINT"   ANZAHL DER ZAHLUNGEN","2
6250 PRINT"   ANZAHL DER JAEHRLICHEN RATEN","3
6260 PRINT"   HOEHE DER RATEN","4
6270 PRINT"   ALLE ANGABEN",,"5
6290 PRINT:INPUT"nUMMER DER GEWUENSCHTEN AENDERUNG EINGEBEN"
;F
6300 IFF<1ORF>5THEN6290
6310 ONFGOTO6020,6040,6030,6050,6000
```

```
6320 F=INT(F):GOTO100
6999 END
13000 CLS: PRINT"EIN FEHLER IST AUFGETRETEN, VERMUTLICH DURC
H
13010 PRINT"FALSCHE DATENEINGABE. WIR MUESSEN NOCH EINMAL
13020 INPUT"VON VORN BEGINNEN. BITTE 'ENTER' DRUECKEN";A$:RU
N
```

Tilgungsdauer

Tilgungsdauer
Das letzte Programm dieser Gruppe ermittelt die Dauer, die man braucht, um bei gegebener Darlehens- und Ratenhöhe sowie bekanntem Zahlungsrhythmus und Zins (Effektivzinssatz!) ein Darlehen zurückzuzahlen. Damit hat man ein weiteres Hilfsmittel zur Kalkulation einer tragbaren Belastung durch einen aufgenommenen Kredit in der Hand.

Auch hier lassen sich für aufeinanderfolgende Rechengänge einzelne Vorgaben ändern. Errechnet wird die Laufzeit in Bruchteilen von Jahren. Um daraus eine Angabe in Jahren und Monaten zu machen, ist eine gesonderte Umrechnung erforderlich. Sie geschieht in den Zeilen 8139, 8140 und 8145.

```
8000 CLS:PRINT"DIESES PROGRAMM ERMITTELD DEN ZEITRAUM, DEN D
IE
8020 PRINT"RUECKZAHLUNG EINES RATENDARLEHENS BEANSPRUCHT.
8030 PRINT"FOLGENDE ANGABEN SIND DAZU ERFORDERLICH:
8035 ONERRORGOTO13000
8050 PRINT:INPUT"HOEHE DES DARLEHENS";OA
8055 IFF=1GOTO8090
8060 INPUT"HOEHE DER RATEN IN DM";PT
8065 IFF=2GOTO8090
8070 INPUT"WIEVIELE ZAHLUNGEN SIND JAEHRLICH ZU LEISTEN";PY
8075 IFF=3GOTO8090
8080 INPUT"WIE HOCH IST DER JAEHRLICHE EFFEKTIVZINSSATZ (%)"
;IR
8090 CLS:IO=OA*IR/100/PY
8095 IFPT<=IOTHENPRINT"UNMOEGLICH! DIESE RATE DECKT NICHT EI
NMAL DIE ZINSEN. DER ZINS  FUER EINE ZAHLUNGSPERIODE BETRAEG
T BEREITS"IO"DM.":GOTO8050
8100 Y=-(LOG(1-(OA*(IR/100))/(PY*PT))/(LOG(1+(IR/100)/PY)*PY
))
8105 NP=Y*PY
8110 PRINT"EIN DARLEHEN UEBER"OA"DM.
8120 PRINT"RUECKZAHLBAR MIT"PY"JAEHRLICHEN RATEN UND
8130 PRINT"MIT EINEM JAEHRLICHEN EFFEKTIVZINSSATZ VON"IR"%
8135 PRINT"ERFORDERT"INT(NP)"RATEN ZU"PT"DM";
8137 IF NP<>INT(NP)PRINT:PRINT"UND EINE RESTZAHLUNG VON";:PR
INTUSING"####.## DM.";PT*(NP-INT(NP))ELSEPRINT".
8139 YP=INT(12*(Y-INT(Y)))+1:IFYP=12YQ=INT(Y)+1ELSEYQ=INT(Y)
8140 PRINT"DIE LAUFZEIT BETRAEGT DAMIT"YQ"JAHRE";
8145 IF YQ=INT(Y) PRINT" UND"YP"MONATE."ELSEPRINT".
8150 PRINT:PRINT"WAS WOLLEN SIE FUER DEN NAECHSTEN RECHENGAN
G AENDERN?
8160 PRINT"1   dARLEHENSSUMME
8170 PRINT"2   HOEHE DER RATEN
8180 PRINT"3   ANZAHL JAEHRLICHER ZAHLUNGEN
8190 PRINT"4   JAHRLICHER EFFEKTIVZINS
8200 PRINT"5   ALLE ANGABEN
8230 PRINT:INPUT"IHRE WAHL";F
8250 IFF<>ABS(INT(F))ORF>5THEN8230
8260 ONFGOTO8050,8060,8070,8080,8000
13000 CLS:PRINT"EIN FEHLER IST AUFGETRETEN, VERMUTLICH INFOL
GE
13010 PRINT"FALSCHER DATENEINGABE. wIR MUESSEN NEU BEGINNEN.
13020 INPUT"BITTE 'ENTER' DRUECKEN";A$:RUN
```

Kreuz und Quer

Kreuz und Quer
Dieses kleine Programm verwendet die graphische Funktion SET(x,y) und zeichnet ohne Unterbrechung immer neue, zufällige Kreuz- und Quer-Muster auf den Bildschirm. Ganz reizvoll zum Zuschauen, zeigt das Programm in einfacher Weise die Einsatzmöglichkeit der Zufallsfunktion RND(x) im Zusammenhang mit der TRS-80-Graphik: Vom Zufall beeinflußt werden Programmschleifen, in denen die SET-Funktionen ausgeführt werden.

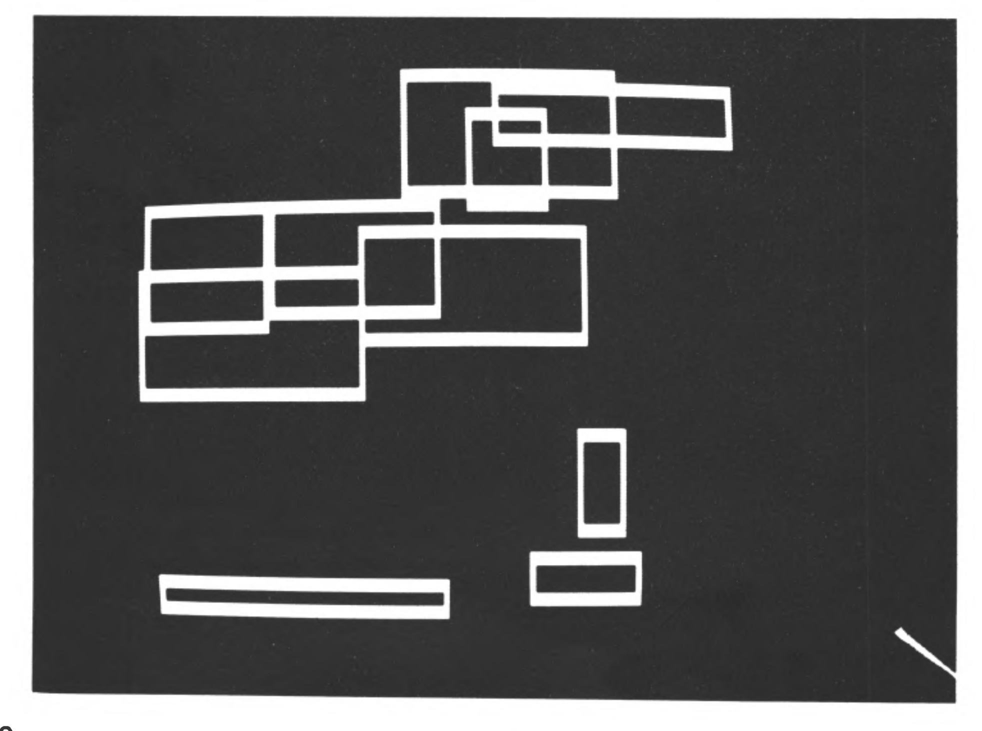
```
3 CLS: DEFINT X,Y
20 FOR X=0 TO 127 STEP RND(10)
30 FOR Y=0 TO 47 STEP RND(5)
40 SET (X,Y)
50 NEXT Y,X
51 GOTO 500
60 FOR X=127 TO 0 STEP -RND(10)
65 FOR Y=47 TO 0 STEP -RND(5)
70 SET (X,Y)
80 NEXT Y,X
85 GOTO 500
90 FOR Y=0 TO 47 STEP RND(10)
100 FOR X=0 TO 127 STEP RND(5)
110 SET (X,Y)
120 NEXT X,Y
125 GOTO 500
130 FOR Y=47 TO 0 STEP -RND(10)
140 FOR X=127 TO 0 STEP -RND(5)
150 SET (X,Y)
160 NEXT X,Y
500 P=RND(6)
510 IF P=1 THEN 20
520 IF P=2 THEN 60
530 IF P=3 THEN 90
540 IF P=4 THEN 130
550 CLS: GOTO 500
```

Rechtecke

Rechtecke
In vielen Einzelheiten ähnelt dieses Programm dem vorherigen. Auch hier werden SET-Funktionen und RND(x) kombiniert, um zufällige Bilder auf den Bildschirm zu malen. Nur handelt es sich dabei nicht um irreguläre Muster, sondern Rechtecke, die gezeichnet werden. Erst wird die Anzahl der zu zeichnenden Rechtecke festgelegt – durch den Zufallswert Q in Zeile 6. Dann folgen für jedes Rechteck Lage (A und D) sowie die beiden Kantenlängen (B und E). In Zeile 50 erfolgt eine Überprüfung, ob ein Rechteck beim Zeichnen über den Bildschirmrand hinausragen würde – wenn ja, wird einfach ein neuer Versuch gestartet. In den beiden Schleifen Zeile 60 bis 80 und 90 bis 120 werden dann zuerst die waagerechten, dann die senkrechten Linien des Rechtecks gezeichnet. Indem in Zeile 130 P solange um 1 erhöht wird, bis es die Größe von Q erreicht, werden soviele Rechtecke gezeichnet, wie der Wert von Q ausmacht. Schließlich sorgt eine Warteschleife in Zeile 140 dafür, daß man sich das entstandene Bild einen Moment lang anschauen kann, und das Spiel beginnt von neuem.

Insgesamt ist es ein recht einfaches Programm. Und doch werden immer neue, optisch reizvolle Bilder auf den Bildschirm gemalt, denen man beim Zuschauen viel Freude abgewinnen kann. Zeile 4 sorgt übrigens für schnelleren Programmablauf, indem das DEFINT A-Z alle verwendeten Zahlen zu ganzen Zahlen erklärt. Die Wirkung dieser Anweisung erkennt man, wenn man das Programm mit RUN 6 startet (anstatt einfach mit RUN). Dann ist Zeile 4 ausgeschaltet, und alles läuft deutlich langsamer ab.

```
4 DEFINT A-Z
6 CLS: Q=RND(2Ø)
8 RANDOM
1Ø A=RND(128)-1
2Ø B=RND(6Ø)
35 D=RND(48)-1
36 E=RND(2Ø)
5Ø IF A+B>127 OR D+E>47 THEN 8
6Ø FOR X=A TO A+B
7Ø SET(X,D)
75 SET(X,D+E)
8Ø NEXT
9Ø FOR Y=D TO D+E
1ØØ SET(A,Y)
11Ø SET(A+B,Y)
12Ø NEXT
13Ø P=P+1
14Ø IF P>=Q FOR I=Ø TO 2ØØØ: NEXT: CLS: P=Ø: Q=RND(2Ø)
15Ø GOTO 8
```



Umwandlung Dezimalzahlen - Hexadezimalzahlen

Umwandlung Dezimalzahlen – Hexadezimalzahlen

Dieses kleine Programm wandelt wahlweise ganze Dezimal- in Hexadezimalzahlen um oder umgekehrt. Die dazu notwendigen Potenzen von 16 werden am Anfang des Programms ermittelt und dann als Variable gespeichert, damit die Berechnungen schneller laufen. Dies geschieht mit Rücksicht auf die für BASIC typische Methode, Zahlen zu verarbeiten, besonders sorgfältig. Andernfalls könnten sich bei den wiederholten Subtraktionen bzw. Multiplikationen kleine Fehler summieren. Berechnet werden bis fünfstellige Hexadezimalzahlen. Dabei werden die Nullen am Anfang der Zahlen nicht unterdrückt, was eigentlich möglich wäre: Diese Zahlen werden oft mit den Nullen geschrieben, besonders bei der Maschinenprogrammierung, bei der diese Nullen ja auch nicht einfach weggelassen werden können. Dezimalzahlen berechnet das Programm bis zur selben Höhe, also bis $16^5 - 1 = 1048575$. Allgemein wurde auch bei diesem Programm auf eine „ordentliche" Bildschirmanzeige geachtet. Durch Eingabefehler kann es nicht so leicht aus dem Konzept gebracht werden. Beides erleichtert die Arbeit mit dem Programm.

```
5 DEFINT P: DEFDBL X,Y: DIM Z$(15): FOR I=0 TO 15: READ Z$(I
): NEXT
7 FOR I=0 TO 4: S(I)=INT(16[I+.2): NEXT: T=16[5: F$="#######
"
10 CLS: PRINT "uMRECHNUNGEN:    DEZ -"CHR$(94)" HEX 1       H
EX -"CHR$(94)" DEZ 2
20 A$=INKEY$: IF A$="2" 480 ELSE IF A$<>"1" 20
80 CLS
100 PRINT@ 0,;: INPUT "DEZIMALZAHL"; X: IF X=0, 10
110 PRINT@ 0, CHR$(30)
115 IF X>=T, 100
120 PRINT@ 320, "DEZIMALZAHL:",,USING F$; X
130 PRINT "HEXADEZIMALZAHL:" TAB(34)
200 FOR I=4 TO 0 STEP -1
210 P=0
220 X=X-S(I)
230 IF X<0 X=X+S(I): PRINT Z$(P);: NEXT: GOTO 100
240 P=P+1: GOTO 220
480 CLS
500 H$="": PRINT@ 0,;: INPUT "HEXADEZIMALZAHL"; H$
510 IF H$="" 10
520 PL=LEN(H$): PRINT@ 0, CHR$(30)
530 IF PL>5, 500
540 PRINT@ 320, "HEXADEZIMALZAHL:" TAB(39-PL)H$
600 Y=0: FOR I=0 TO PL-1
630 FOR J=0 TO 15
650 IF MID$(H$,PL-I,1)=Z$(J) Y=Y+J*S(I): NEXT I: GOTO 700
670 NEXT J: GOTO 480
700 PRINT "DEZIMALZAHL:",,USING F$; Y: GOTO 500
1000 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
```

Ein-und zweidimensionale Matrix

Ein- und zweidimensionale Matrix
Wenn man diese beiden kleinen Programme laufen läßt, kommt jeweils das gleiche dabei heraus: 35 verschiedene Zahlen, schön ordentlich in fünf Reihen zu je sieben Zahlen auf den Bildschirm gebracht. Zweck dieser Übung ist zu zeigen, daß man mit einem kleinen Kunstgriff auch in einer eindimensionalen Matrix enthaltene Zahlen zweidimensional ausgeben kann, d. h., in Form eines Rechtecks mit passender Zeilen- und Spaltenzahl. Das Geheimnis liegt in der Zeile 50 des ersten Programms. Sie teilt die 35 Elemente der Matrix A(35) in fünf Gruppen zu je sieben Zahlen auf, die dann jeweils in einer Zeile ausgedruckt werden. Im zweiten Programm, das mit der zweidimensionalen Matrix A(5,7) arbeitet, ist das nicht notwendig.

Gebrauchen kann man diesen Programmiertrick zum Beispiel in Programmen, in denen aus anderen Gründen Zahlen in Form einer eindimensionalen Matrix vorliegen und man sie ohne große Umstände als rechteckigen Block auf den Bildschirm bringen will.

Übrigens: Die Anweisung DIM A(35) des ersten Programms ist im zweiten nicht erforderlich und fehlt deshalb. Der Computer ordnet ja jeder Matrix automatisch 11 Elemente pro Dimension zu (0 bis 10), und das A(x,y) geht ja nur bis A(5,7).

```
10 'EINDIMENSIONALE MATRIX
20 DIM A(35)
30 FOR I=1 TO 5
40 FOR J=1 TO 7
50 N=7*(I-1)+J
60 A(N)=I+J/10
70 PRINT A(N);
80 NEXT J
90 PRINT
100 NEXT I
110 END
210 'ZWEIDIMENSIONALE MATRIX
220 FOR I=1 TO 5
230 FOR J=1 TO 7
240 A(I,J)=I+J/10
250 PRINT A(I,J);
260 NEXT J
270 PRINT
280 NEXT I
290 END
```

Geburtstage am gleichen Tag

Geburtstage am gleichen Tag

Wer hätte gedacht, daß von 40 zufällig ausgewählten Personen mit fast 90 %iger Wahrscheinlichkeit zwei am gleichen Tag Geburtstag haben? Nun, dieses kurze Programm erstellt eine Liste dieser Wahrscheinlichkeiten für Personengruppen von 2 bis 40 Personen. Eine Möglichkeit der Überprüfung der Angaben ergibt sich zum Beispiel auf einer Party, wo eine mehr oder weniger bunt zusammengewürfelte Personengruppe beisamen ist.

In Zeile 470 findet sich eine PRINT USING "..."-Anweisung, die sehr schön die vorteilhafte Anwendungsmöglichkeit dieses Formatierungsbefehls zeigt. Die einzelnen Zahlen werden automatisch auf zwei Stellen hinter dem Komma gerundet, und ein Prozentzeichen wird mit einer Stelle Zwischenraum nachgestellt. Zusätzlich sind die Zahlen auf den Dezimalpunkt ausgerichtet. Man sieht auch, daß vor dem USING in derselben PRINT-Anweisung ruhig noch andere, zu druckende Ausdrücke stehen dürfen, auch Dinge wie TAB(x). Dagegen muß man mit der Ausdehnung der PRINT-Anweisung nach dem USING Vorsicht walten lassen — hier sind nur wenige Möglichkeiten gegeben.

```
410 PRINT"ANZAHL DER          WAHRSCHEINLICHKEIT, DASS MINDEST
ENS
420 PRINT"PERSONEN            ZWEI AM GLEICHEN TAG GEBOREN SIN
D
430 PRINT"----------          --------------------------------
---
440 Q=364/365
450 FOR N=2 TO 40
460 P=100*(1-Q)
470 PRINT"   "NTAB(25)USING "##.## %";P
480 Q=Q*(365-N)/365
490 NEXT
```

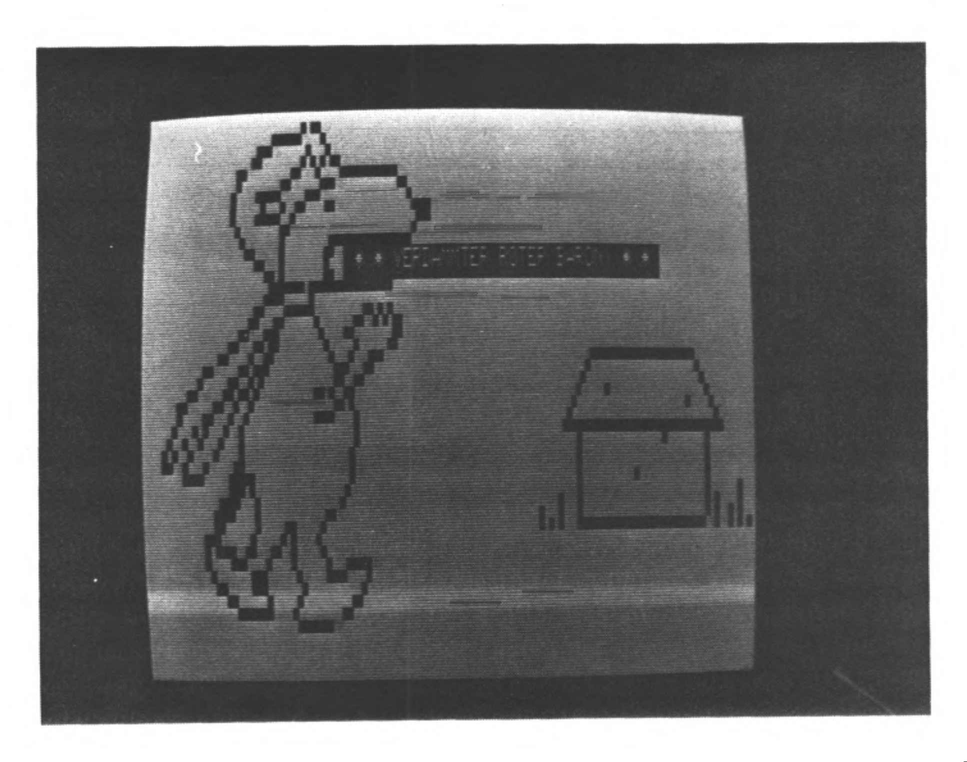
Snoopy

Snoopy
Die meisten Leser werden sicher von Snoopy gehört haben, dem merkwürdigen Hund aus den Comic Strips des amerikanischen Zeichners Charles M. Schulz. Snoopy hat eine sehr lebendige Phantasie und träumt davon, ein berühmtes Flieger-As im ersten Weltkrieg zu sein, der sich mit dem deutschen Flieger-As Baron von Richthofen — bekannt als der "Rote Baron" — aufregende Luftkämpfe liefert. Davon handelt auch dieses Programm: Es wird die Figur des Snoopy schwarz auf weiß gezeichnet, der mit einer Fliegermontur bekleidet sein soll, sowie seine Hundehütte, die in seiner Phantasie sein Flugzeug darstellt. Die Hundehütte wird getroffen, worauf Snoopy ausruft: „Verdammter Roter Baron!" Dies wiederholt sich zehnmal, dann erscheint die ganze Zeichnung noch einmal als Negativ, Snoopy verwünscht noch einmal seinen Gegner und das Programm beginnt von vorn.

Allzu faszinierend ist das ganze wohl nicht gerade, doch wurde dieses Programm eigentlich auch aus einem anderen Grund in dieses Buch aufgenommen: Hier kann man sehr schön sehen, wie sich Daten mit Hilfe der READ-Anweisung und DATA-Zeilen in einem Programm raumsparend unterbringen und auch gliedern lassen. Aufgabe ist es ja, eine Zeichnung auf den Bildschirm des Computers zu bringen. Die einfachste Lösung wäre es, für jeden Bildpunkt die jeweiligen waagerechten und senkrechten Koordinaten, die für die SET- bzw. RESET-Anweisung nötig sind, in den DATA-Zeilen zu speichern, so daß sie durch READ-Befehle übernommen und in das gewünschte Bild umgewandelt werden können. Dazu wären pro Bildpunkt also zwei Zahlen erforderlich — ein ziemlicher Aufwand. Dieses Programm kommt mit wesentlich weniger Daten aus, indem es das Bild zunächst zeilenweise aufbaut, d. h. jeweils alle Punkte mit derselben senkrechten Koordinate zusammenfaßt und diese Koordinate nur einmal angibt. Darüberhinaus wer-

den von waagerechten Linien nicht alle Punkte, sondern nur Anfangs- und Endpunkte angegeben. Um aus diesen Angaben das Bild aufzubauen, muß der Computer bei den Zahlen der DATA-Zeilen waagerechte und senkrechte Koordinaten unterscheiden können und außerdem normale Bildpunkte von den Anfangs- und Endpunkten der waagerechten Linien trennen. Dieses Problem ist folgendermaßen gelöst:

Die senkrechte Koordinate aller auf einer Höhe liegenden Bildpunkte, die gleichbedeutend ist mit der jeweiligen Zeilennummer, wird nicht in ihrer eigentlichen Form im Bereich 0 bis 47 angegeben, sondern als negative Zahl. Auch die Anfangs- und Endpunkte der waagerechten Linien erscheinen nicht in ihrer wahren Gestalt, sondern durch 1000 dividiert als Dezimalbrüche. Daran kann der Computer diese verschiedenen Zahlen erkennen – das geschieht in den beiden Zeilen 16220 und 16230 – und nach entsprechender Umformung in richtiger Weise einsetzen. In den Zeilen 16430 und 16440 wird der Vorgang für das Negativbild wiederholt.



Auch das Ende des READ-Vorgangs muß signalisiert werden, da sich die Anweisungen dafür in einer endlosen Schleife befinden. Dazu dient die Kennzahl 3500, die das Ende der unter DATA zusammengefaßten Daten bildet. Trifft der Computer auf diese Zahl, bricht er den Lese- und RESET- bzw. SET-Kreislauf ab und fährt mit dem Programm fort (Zeile 16210 bzw. 16420).

Man sieht also, auch in den DATA-Zeilen lassen sich außer den eigentlichen Zahlen auch noch weitere, für das Programm wichtige Informationen unterbringen, die in diesem Fall dazu dienen, die Menge der nötigen Punktkoordinaten zum Zeichnen eines Bildes zu verringern. Selbstverständlich lassen sich auch noch viele andere Anwendungen derartiger Datenorganisation denken — auch bei der Speicherung von Strings in DATA-Zeilen.

```
16000 'COPYRIGHT 1978  THE BOTTOM SHELF INC.
16010 'ALL RIGHTS RESERVED    DO NOT COPY
16020 'P.O. BOX 49104  ATLANTA  GA   30359
16200 CLS:CLEAR500:A$=STRING$(63,CHR$(191)):FORX=1TO15:PRINT
A$:NEXTX:PRINTA$;
16210 READA:IFA=3500THENGOTO16300
16220 IFA<0THENY=-A:GOTO16210
16230 IFA<1THENB=A*1000:READA:C=A*1000:FORX=BTOC:RESET(X,Y):
NEXTX:GOTO16210
16240 X=A:RESET(X,Y):GOTO16210
16300 FORX=1TO10:RESET(111,25):RESET(106,28):RESET(100,31):R
ESET(94,24):FORS=1TO300:NEXTS:PRINT@276," * * VERDAMMTER ROT
ER BARON! * * ";:FORS=1TO300:NEXTS:T$=STRING$(33,CHR$(191)):
PRINT@276,T$;:SET(111,25):SET(106,28):SET(100,31):SET(94,24)
16350 FORS=1TO300:NEXTS,X
16400 FORX=1TO1000:NEXTX:CLS
16410 RESTORE
16420 READA:IFA=3500THENFORX=1TO1000:NEXTX:GOTO16500
16430 IFA<0THENY=A*-1:GOTO16420
16440 IFA<1THENB=A*1000:READA:C=A*1000:FORX=BTOC:SET(X,Y):NE
XTX:GOTO16420
16450 X=A:SET(X,Y):GOTO16420
16500 PRINT@275," * * VERDAMMTER ROTER BARON! * * ";:FORX=1T
```

```
O2ØØØ:NEXTX:GOTO16ØØØ
169ØØ DATA-1,.Ø32,.Ø34,-2,.Ø26,.Ø31,35,-3,.Ø23,.Ø25,32,36,-4
,.Ø21,.Ø23,31,33,35,37,38,-5,18,19,29,3Ø,34,.Ø39,.Ø5Ø,-6,18,
27,28,35,36,37,51,52,-7,17,.Ø23,.Ø27,32,33,34,53
1691Ø DATA-8,17,23,27,3Ø,31,36,37,.Ø54,.Ø57,-9,17,.Ø23,.Ø27,
29,.Ø55,.Ø57,-1Ø,18,27,28,54,-11,19,27,.Ø37,.Ø53,-12,2Ø,21,2
7,36,37,-13,22,23,27,36,-14,24,25,27,36,37,-15,.Ø26,.Ø49
1692Ø DATA-16,24,25,26,27,33,-17,22,23,.Ø28,.Ø33,44,45,47,49
,-18,21,22,26,27,34,42,43,46,48,5Ø,51,-19,19,2Ø,24,25,26,35,
4Ø,41,5Ø,51,-2Ø,17,18,22,23,25,26,36,42,49,5Ø,-21,15,16,21,2
4,25,25,37,41,45,46,47,.Ø91,.113
1693Ø DATA-22,12,13,19,2Ø,22,23,24,38,39,4Ø,44,45,9Ø,113,-23
,11,17,18,21,23,38,39,42,43,89,114,-24,9,1Ø,16,17,19,2Ø,22,3
4,35,37,4Ø,41,88,115
1694Ø DATA-25,7,8,14,15,18,21,38,39,41,87,116,-26,7,12,13,17
,2Ø,.Ø34,.Ø37,42,86,117,-27,6,1Ø,12,16,19,42,.Ø85,.118,-28,5
,8,11,15,19,42,88,115
1695Ø DATA-29,4,6,14,9,19,41,88,115,-3Ø,4,5,8,12,18,19,41,88
,115,-31,8,9,1Ø,11,.Ø17,.Ø21,41,88,115,-32,17,18,19,21,4Ø,88
,115,122,-33,15,17,16,18,22,39,84,88,115,117,122
1696Ø DATA-34,14,15,16,17,22,36,37,8Ø,84,88,115,117,12Ø,122,
-35,14,15,22,28,29,35,8Ø,82,84,.Ø88,.115,117,12Ø,122,124,-36
,12,13,14,21,27,29,34,-37,12,.Ø14,.Ø17,2Ø,21,25,26,29,35
1697Ø DATA-38,12,17,18,19,24,29,36,.Ø4Ø,.Ø44,-39,13,21,22,23
,3Ø,.Ø36,.Ø39,44,-4Ø,14,15,21,22,23,31,43
1698Ø DATA-41,16,17,24,3Ø,41,42,-42,.Ø18,.Ø24,29,39,4Ø,-43,.
Ø3Ø,.Ø37,35ØØ
```

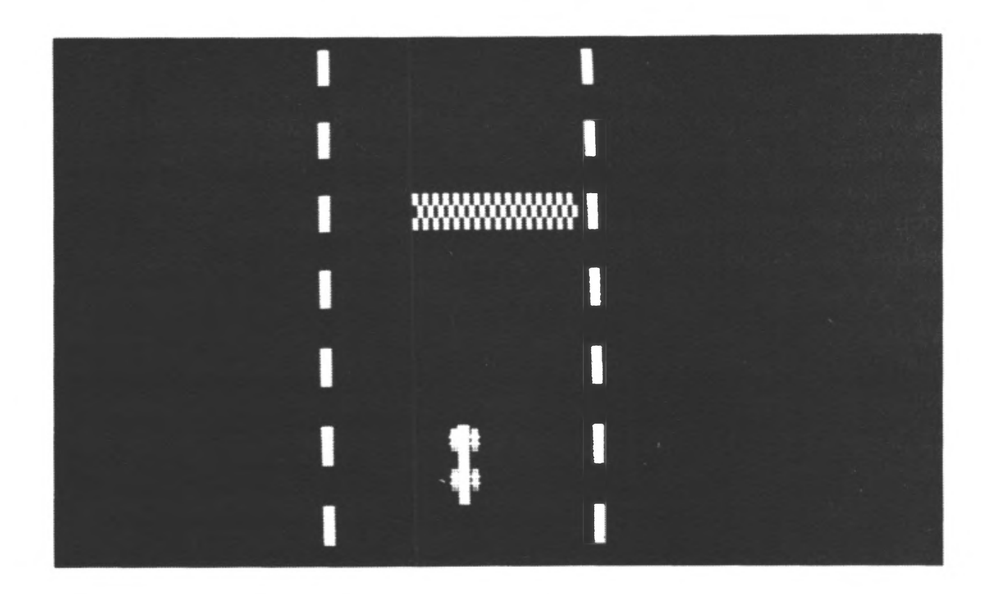
Autorennen

Dieses Programm erzeugt auf dem Bildschirm des TRS-80 ein Rennwagenspiel ähnlich denen, die man von den TV-Spielgeräten her kennt. Ein Rennwagen fährt auf einer Straße und in regelmäßigen Abständen tauchen Hindernisse auf, denen das Auto ausweichen muß. Gelingen diese Ausweichmanöver, ohne daß man auf die Fahrbahnbegrenzungen gerät, erhöht sich die Fahrtgeschwindigkeit in zwei Stufen. Bei jeder Karambolage wird wieder auf das langsamste Tempo zurückgeschaltet. Am Anfang werden auch noch die Steuerbefehle für den Rennwagen gezeigt:"Z" für "nach links", "/" für "nach rechts" und die Leertaste für „geradeaus".

Auf anschauliche Art und Weise zeigt dieses Programm die graphischen Möglichkeiten, die der TRS-80 zusammen mit den Zeichenketten bietet. Wie bei einem Zeichentrickfilm sind zum Beispiel von den Fahrbahnbegrenzungen jeweils sechs Bewegungsphasen durch Zusammenfügen von graphischen Zeichen und Cursor-Kontrollfunktionen zu einzelnen Strings vorhanden (Z(0) bis Z(5)). Sie werden nacheinander durch einfache PRINT-Befehle gezeigt und vermitteln dadurch den Eindruck der Bewegung. Für die Hindernisse gibt es insgesamt 48 verschiedene Zeichenketten, die in ansteigender Länge nacheinander erscheinen und ein sich nach unten auf dem Bildschirm bewegendes Hindernis ergeben, obwohl sie immer an derselben Stelle am oberen Bildschirmrand anfangen: Vor den eigentlichen, sichtbaren graphischen Zeichen werden immer mehr „neue Zeilen-Anweisungen" (Codezahl: 26) geschaltet. Schließlich ist auch der Rennwagen selbst eine einzelne Zeichenkette, zusammengesetzt aus acht String-Komponenten (Zeile 100).

Das ganze, aufwendige und auch speicherraumzehrende Zeichenketten-Zusammenfügen hat den Sinn, beim tatsächlichen Spielablauf mit so-

wenig PRINT-Anweisungen wie nur möglich auszukommen und diesen Teil des Programms auch sonst möglichst einfach zu gestalten, damit eine einigermaßen attraktive Fahrgeschwindigkeit des Autos dabei herauskommt. Leider zeigen sich hier aber trotzdem deutlich die Grenzen eines Computers mit BASIC-Interpreter, der ja das Programm Zeile für Zeile erst übersetzt und dann ausführt.



Besser geeignet wäre eigentlich ein entsprechendes Maschinenprogramm — wie bei allen Vorgängen, bei denen es in erster Linie auf Schnelligkeit ankommt. Immerhin gibt das Programm einen guten Eindruck von den eleganten Programmiertechniken, die mit dem Level II-BASIC auf graphischem Gebiet möglich sind.

Noch zwei Anmerkungen: Will man das Spiel etwas schwieriger machen, kann man die Breite des Hindernisses vergrößern ; sie verbirgt sich hinter der Zahl 16 in den Zeichenketten ZH(0,0) bis ZH(5,0). Dann muß man allerdings auch entsprechend den Spielraum der möglichen Positionen des Hindernisses verändern (RND10) in Zeile 500) sowie auch die Bedingungen für einen "Unfall" des Rennautos, die sich in der Zeile 540 befinden.

Die erste dieser Bedingungen (von links) bedeutet „Anstoß links", die zweite „Anstoß rechts" und die drei übrigen zusammen „Zusammenstoß mit dem Hindernis". Die in dem Programm gewählte Schreibweise ist dabei wie schon an anderer Stelle ausgeführt, nicht die einzig mögliche. Es geht auch so: IF (P< =658)+(P>=682)+(J=5)*(P>=PR+638) *(P< = PR+655) THEN 1000.

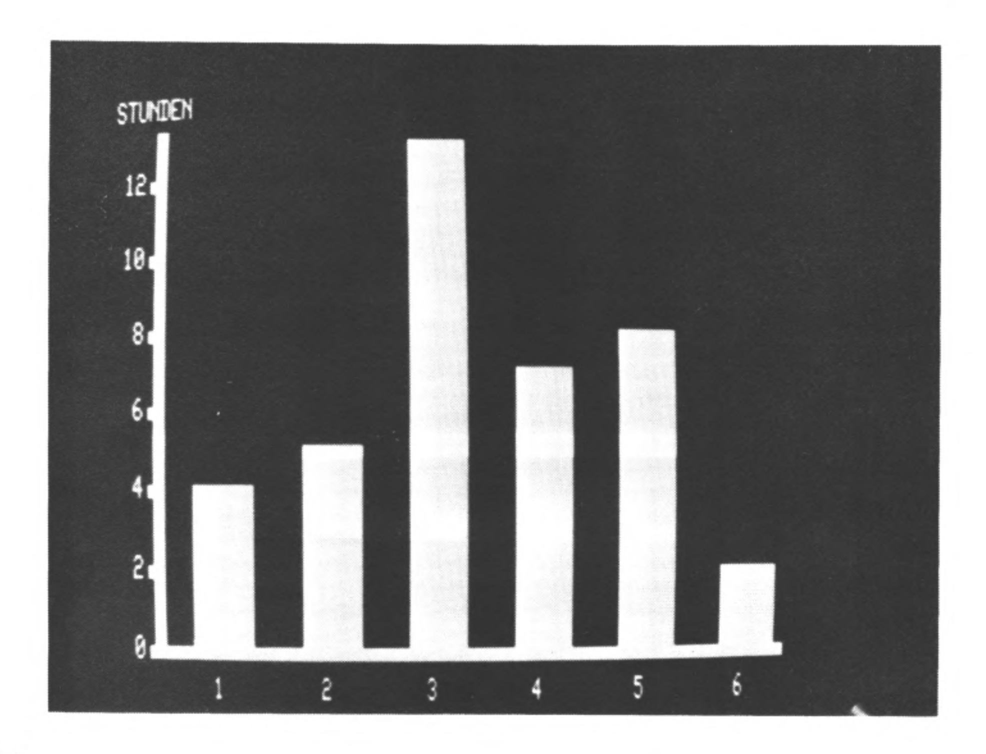
Dabei ist zu beachten, daß wie bei gewöhnlichen Rechenausdrücken auch bei solchen logischen Ausdrücken Punktrechnung vor Strichrechnung geht, also "*" vor "+".

```
1 'COPYRIGHT 1979 STUEBS
3 CLS: CLEAR 5000
5 PRINT@15, "* * *  AUTORENNEN  * * *
7 PRINT:PRINT:PRINT"'Z' = NACH LINKS":PRINT:PRINT"'/' = NACH
 RECHTS":PRINT:PRINT"LEERTASTE = GERADEAUS
10 DEFSTR Z: DEFINT I,J,M,P,S
15 ZS=CHR$(136)+STRING$(2,153)+CHR$(26)+STRING$(3,8)+CHR$(13
6)+STRING$(2,153)
20 D$=CHR$(26): C$=CHR$(27): B$=CHR$(8): BB$=STRING$(16,8)
25 ZA=CHR$(131)+D$+B$: ZB=CHR$(143)+D$+B$: ZC=CHR$(188)+D$+B
$: ZD=CHR$(176)+D$+B$: ZE=CHR$(191)+D$+B$+CHR$(128)+D$+B$
30 Z(0)=ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+CHR$(18
8)
35 Z(1)=ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+CHR$(17
6)
40 Z(2)=ZE+ZE+ZE+ZE+ZE+ZE+ZE+CHR$(191)+D$+B$
45 Z(3)=ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+ZA+ZC+CHR$(13
1)
50 Z(4)=ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+ZB+ZD+CHR$(14
3)
55 Z(5)=CHR$(128)+D$+B$+ZE+ZE+ZE+ZE+ZE+ZE+ZE+CHR$(191)
90 P=670: S=1
100 ZZ="#"+CHR$(191)+"#"+CHR$(26)+STRING$(3,8)+"#"+CHR$(191)
+"#"
110 ZO="   "+CHR$(26)+STRING$(3,8)
200 ZH(0,0)=STRING$(16,129)+C$+BB$+STRING$(16,152)+C$+BB$
205 ZH(1,0)=STRING$(16,137)+C$+BB$+STRING$(16,144)+C$+BB$
210 ZH(2,0)=STRING$(16,153)+C$+BB$+STRING$(16,128)+C$+BB$
215 ZH(3,0)=STRING$(16,152)+D$+BB$+STRING$(16,129)+C$+C$+BB$
```

```
220 ZH(4,0)=STRING$(16,144)+D$+BB$+STRING$(16,137)+C$+C$+BB$
225 ZH(5,0)=STRING$(16,128)+D$+BB$+STRING$(16,153)+C$+C$+BB$
300 FOR I=0 TO 5
310 FOR J=1 TO 7
320 ZH(I,J)=STRING$(J*2,26)+ZH(I,0)
330 NEXT J,I
350 CLS
500 PR=18+RND(10): FOR J=0 TO 7: PRINT@ 979, "
            ";: FOR I=S-1 TO 5 STEP S: PRINT@ 18, Z(I);: PRIN
T@ 44, Z(I);: PRINT@ PR, ZH(I,J);
520 ZI=INKEY$: IF ZI="Z" M=-S ELSE IF ZI="/" M=S ELSE IF ZI=
" " M=0
540 PRINT@ P, ZO;: P=P+M: PRINT@ P, ZZ;: IF P<=658 OR P>=682
 OR (J=5)+(P>=PR+638)+(P<=PR+655)=-3 THEN 1000
550 I1=I1+1: IF I1=100 S=2 ELSE IF I1=200 S=3
600 NEXT I,J: GOTO 500
1000 FOR I2=1 TO 50: PRINT@ P, ZS;: PRINT@ P, ZZ;: NEXT
1010 M=0: I1=0: S=1: IF P<=658 P=659 ELSE IF P>=682 P=681
1020 CLS: GOTO 600
```

Balkengraphik

Balkengraphik
Dieses Programm zeichnet eine graphische Darstellung von eingegebenen Werten als Balkengraphik auf den Bildschirm des Computers. Dabei können Anzahl der Balken, Bezeichnungen und Skala der senkrechten y-Achse innerhalb gewisser Grenzen frei gewählt werden. Richtige Anordnung der gewünschten Wörter, Skaleneinteilung und Verteilung der Balken auf den zur Verfügung stehenden Raum besorgt das Programm automatisch. Für die y-Achse gibt es vier mögliche Einteilungen: 3, 4, 6 und 13 Teilstriche – entsprechend der 16-zeiligen Bildschirmaufteilung. Für die zugehörigen Zahlen stehen 2 Stellen (und eine Stelle für eventuelle Minuszeichen) zur Verfügung. Für Dezimalbrüche ist damit im allgemeinen kein Platz, was man bei der Festlegung des Wertebereichs berücksichtigen muß.



WAS WILLST DU AENDERN?

1 - EINHEIT X-ACHSE
2 - EINHEIT Y-ACHSE
3 - BALKENZAHL
4 - BALKENBEZEICHNUNGEN
5 - UNTERE GRENZE DER Y-WERTE
6 - OBERE GRENZE DER Y-WERTE
7 - EINTEILUNG DER Y-ACHSE
8 - BALKENWERTE
9 - ALLES KLAR

BITTE GEWUENSCHTE ZAHL EINGEBEN? _

Ein wenig mehr Platz hat man sich für die Zahlen an der senkrechten Achse geschaffen, indem man in den Zeilen 230 und 310 jeweils den Ausdruck USING " # # # "; in die PRINT-Anweisung einfügte. Dann sind zwar Brüche ausgeschlossen, aber man kriegt auch 3-stellige Zahlen unter, allerdings ohne Minuszeichen. Darüberhinaus sind Zahlen mit verschiedener Stellenzahl dann richtig ausgerichtet. Und man braucht sich auch nicht mehr darum zu kümmern, ob die Achseinteilung ganze Zahlen ergibt – die USING-Funktion sorgt ja gleichzeitig für die Rundung der Zahlen auf das angegebene Format.

Die Zahl der Balken ist auf 12 begrenzt. Theoretisch wären auch noch mehr möglich, aber die Aufteilung der Balken auf dem Bildschirm würde bei entsprechend geändertem Programm nicht mehr so schön klappen. Das Programm fragt am Anfang auch nach Bezeichnungen für x- und y-Achse (waagerecht und senkrecht). Auf den Namen für die x-Achse muß man bei mehr als 3–4 Balken im allgemeinen verzichten (durch einfaches „ENTER"-Drücken), weil er neben den Balkennamen keinen Platz mehr hat. Überhaupt müssen diese Bezeichnungen

natürlich um so kürzer werden, je mehr Balken man verwendet. Da die verschiedenen Wörter in der Graphik außerdem Zeichenketten (Strings) darstellen, kann einem bei vielen und langen Bezeichnungen der vom Computer automatisch reservierte String-Speicherraum ausgeben, und er reagiert mit der Fehlermeldung ?OS ERROR IN (Zeilennummer). Dem kann ein CLEAR 100, irgendwo am Anfang des Programms eingefügt, abhelfen.

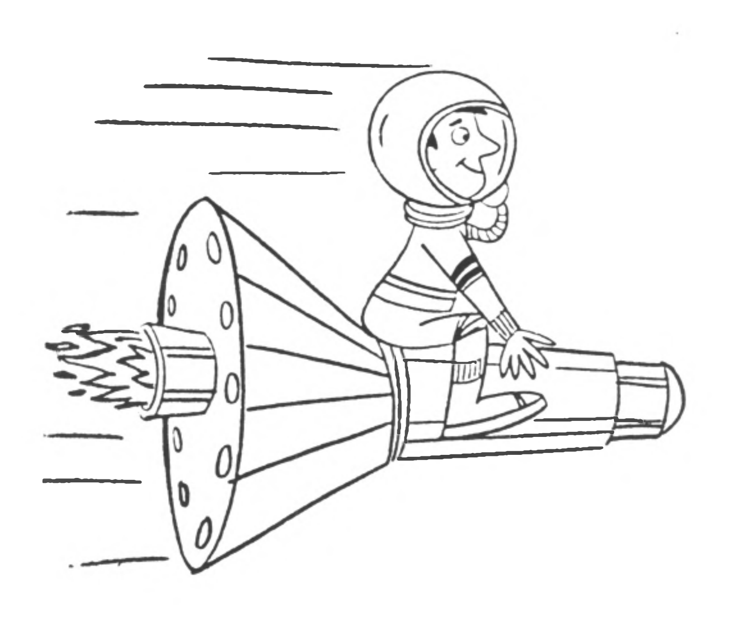
Das Programm läßt sich gut einsetzen, wenn man irgendwelche graphischen Darstellungen benötigt. Dafür kann man dann den Bildschirm abfotographieren. Auch läßt sich das Programm vorteilhaft in andere Programme einbauen, wenn man eine solche Darstellung darin auftretender Werte wünscht. Dazu müssen allerdings die Programmzeilen entsprechend abgeändert werden, in denen Angaben vom Benutzer verlangt werden. Diese Werte soll ja nun das übergeordnete Programm liefern.

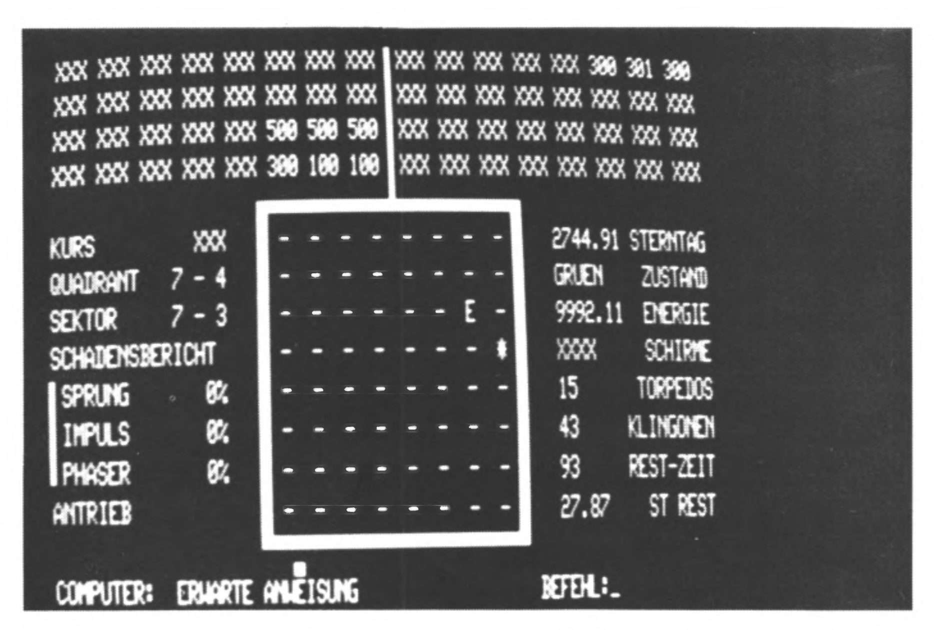
```
10 CLS
15 DIM BW(12), BN$(12)
20 PRINT"BALKENGRAFIK
25 PRINT"------------
30 PRINT: INPUT"EINHEIT X-ACHSE";X$
40 INPUT"EINHEIT Y-ACHSE";Y$
50 INPUT"ANZAHL DER BALKEN";BZ
60 PRINT"BEZEICHNUNG DER EINZELNEN BALKEN:
70 FOR I=1 TO BZ
80 PRINT"BALKEN NR."I;: INPUT BN$(I)
90 NEXT
100 INPUT"UNTERE GRENZE DER Y-WERTE";YU
110 INPUT"OBERE GRENZE DER Y-WERTE ";YO
120 INPUT"EINTEILUNG DER Y-ACHSE (ANZAHL DER tEILSTRICHE)";YT
125 GOSUB 4000
130 PRINT"WERTE FUER DIE EINZELNEN BALKEN:
140 FOR I=1 TO BZ
150 PRINT"BALKEN NR."I;: INPUT BW(I)
160 NEXT
190 CLS
200 YD=YO-YU
210 PRINT@1023-LEN(X$),X$;
220 PRINT@0,Y$
230 PRINT@896, USING "###"; YU;
```

```
250 IF YT=13 M=1 ELSE M=12/YT
300 FOR I=1 TO YT
310 PRINT@ 896-I*64*M, USING "###"; YU+I*YD/YT;
320 NEXT
500 FOR Y=3 TO 42: SET(8,Y): SET(9,Y): NEXT
530 FOR X=7 TO 127: SET(X,43): NEXT
560 FOR I=1 TO YT: SET(7,43-3*I*M): NEXT
600 XD=60/BZ
610 FOR I=1 TO BZ
620 PRINT@965+XD/2+(I-1)*XD-LEN(BN$(I))/2,BN$(I);
630 NEXT
700 FOR I=1 TO BZ
710 FOR X=10+XD/2+2*(I-1)*XD TO 10+3*XD/2+2*(I-1)*XD
720 IF M<>1 GOTO 740
730 FOR Y=43-39*(BW(I)-YU)/YD TO 43: GOTO 750
740 FOR Y=43-36*(BW(I)-YU)/YD TO 43
750 SET(X,Y)
760 NEXT Y,X,I
1000 S$=INKEY$: IF S$="" THEN 1000
1100 CLS: PRINT"WAS WILLST DU AENDERN?
1110 PRINT: PRINT"1 - EINHEIT X-ACHSE
1120 PRINT"2 - EINHEIT Y-ACHSE
1130 PRINT"3 - BALKENZAHL
1140 PRINT"4 - BALKENBEZEICHNUNGEN
1150 PRINT"5 - UNTERE GRENZE DER Y-WERTE
1160 PRINT"6 - OBERE gRENZE DER Y-WERTE
1170 PRINT"7 - EINTEILUNG DER Y-ACHSE
1180 PRINT"8 - BALKENWERTE
1190 PRINT"9 - ALLES KLAR
1200 PRINT: INPUT"BITTE GEWUENSCHTE ZAHL EINGEBEN";W
1210 CLS: ON W-1 GOTO 1400,1500,1600,1700,1800,1900,2000,190
1300 PRINT"EINHEIT X-ACHSE: "X$;:INPUT" - NEU";X$: GOTO 1100
1400 PRINT"EINHEIT Y-ACHSE: "Y$;:INPUT" - NEU";Y$: GOTO 1100
1500 PRINT"BALKENZAHL:"BZ;:INPUT" - NEU";BZ: GOTO 1100
1600 FOR I=1 TO BZ
1610 PRINT"BALKENNAME NR."I":"BN$(I);:INPUT" - NEU";BN$(I)
1620 NEXT: GOTO 1100
1700 PRINT"UNTERE GRENZE DER Y-WERTE:"YU;:INPUT" - NEU";YU: G
OTO 1100
1800 PRINT"OBERE GRENZE DER Y-WERTE:"YO;:INPUT" - NEU";YO: GO
TO  1100
1900 PRINT"EINTEILUNG DER Y-ACHSE:"YT;:INPUT" - NEU";YT: GOSU
```

```
B 4000: GOTO 1100
2000 FOR I=1 TO BZ
2010 PRINT"BALKENWERT NR."I":"BW(I);:INPUT" - NEU";BW(I)
2020 NEXT: GOTO 1100
4000 IF (YT=3)+(YT=4)+(YT=6)+(YT=13)=0 INPUT"NUR 3, 4, 6 ODER
 13 TEILSTRICHE, BITTE! - NOCHMAL";YT: GOTO 4000
4010 RETURN
```

Startrek




XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX 300 301 300
XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX
XXX XXX XXX XXX XXX 500 500 500 XXX XXX XXX XXX XXX XXX XXX XXX
XXX XXX XXX XXX XXX 300 100 100 XXX XXX XXX XXX XXX XXX XXX XXX
KURS XXX
QUADRANT 7 - 4
SEKTOR 7 - 3
SCHADENSBERICHT
SPRUNG 0%
IMPULS 0%
PHASER 0%
ANTRIEB
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - E -
- - - - - - - - - *
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
2744.91 STERNTAG
GRUEN ZUSTAND
9992.11 ENERGIE
XXXX SCHIRME
15 TORPEDOS
43 KLINGONEN
93 REST-ZEIT
27.87 ST REST
COMPUTER: ERWARTE ANWEISUNG
BEFEHL:_

Startrek
Dieses Programm ist unter Computerfachleuten eines der beliebtesten Spiele. Es existiert in vielen Varianten und wird vielfach auch vom Bedienungspersonal großer, kommerzieller Computer gespielt. Diese Version ist auf den TRS-80 zugeschnitten und weist einige reizvolle Besonderheiten auf, die man nicht oft findet. Als ungewöhnlich umfangreiches Programm füllt es den freien Speicherraum eines TRS-80 mit 16-K-RAM fast vollständig aus.

„Startrek" ist der amerikanische Name für die Science-Fiction-Fernsehserie, die in der Bundesrepublik unter dem Namen „Raumschiff Enterprise" bekannt ist und in der Commander Kirk, Mr. Spock und all die anderen Helden vielfältige Weltraumabenteuer zu bestehen haben. Dies ist auch das Thema des Spiels – als Commander der Enterprise fliegt man durch den Weltraum und muß die Raumschiffe der feindlichen Klingonen aufspüren und vernichten. Wenn einem das gelingt, hat man die Galaxie gerettet und wird (hoffentlich!) zum Admiral befördert. Andernfalls geht die Enterprise verloren – und damit auch das Spiel.

Nach Start des Programms erscheint für kurze Zeit die Mitteilung „Instrumente angeschaltet – Kommando-Übernahme in 4 Sek." und dann – als eigentliches Objekt des Spiels – der Kommandostand des Raumschiffs.

Das Monitorbild enthält zunächst einmal Informationen über die nähere und weitere Umgebung der Enterprise: Der Weltraum, in dem das Raumschiff operiert, ist in 64 Quadranten zu je 64 Rektoren eingeteilt. Im Zentrum des Bildschirms ist der Quadrant abgebildet, in dem sich die Enterprise gerade befindet. Die 64 Sektoren dieses Quadranten bilden eine 8 x 8-Matrix; darin erscheint die Enterprise als E, Sterne als * und feindliche Klingonen-Schiffe, kurz Klingonen genannt, als K. Es gibt auch einige wenige Raumstationen, die die Enterprise aufsuchen kann, um Energie nachzutanken und Reparaturen ausführen zu lassen. Sie werden als ! dargestellt. Leere Sektoren erscheinen als –.

Sämtliche 64 Quadranten des Operationsgebietes finden sich im oberen Teil des Bildschirms – zunächst jeweils als XXX. Für beide An-

ordnungen, Quadranten sowie Sektoren des einen Quadranten, der in der Bildschirmmitte dargestellt ist, gilt ein 8 x 8 - Koordinatensystem, das in der linken oberen Ecke beginnt, dort also die Position 1,1 hat. Dabei sind die oberen 16 Quadranten links auf dem Bildschirm zu finden, die unteren 16 rechts.

Des weiteren finden sich Informationen über den Zustand des Raumschiffes auf dem Monitor: Links oben neben dem zentralen Quadrantenbild zunächst der Kurs, den das Raumschiff gerade fliegt oder zuletzt geflogen ist, als dreistellige Zahl in der üblichen Gradeinteilung, nämlich 000 für senkrecht nach oben, 090 für nach rechts, 180 für abwärts und 270 für nach links (sowie natürlich auch alle Zwischenwerte). Darunter die Position der Enterprise in Quadranten- und Sektoren-Koordinaten. Dann folgt der sogenannte Schadensbericht – Angaben über Schäden, die die beiden Antriebe „Sprung" und „Impuls" sowie die „Phaser"-Strahlwaffe im Kampf mit den Klingonen davongetragen haben (ausgedrückt in %). Schließlich als letzte Angabe auf der linken Seite des Bildschirms die Antriebsart, die gerade verwendet wird, bzw. wurde. Rechts oben das Datum, als Sterntag angegeben (ohne Bedeutung für das Spiel), dann der Zustand des Schiffes: Hier gibt es drei Möglichkeiten – ROT, wenn sich Klingonen im selben Quadranten mit der Enterprise befinden, GRÜN, wenn keine Klingonen in der Nähe sind und DOCK, wenn die Enterprise an einer Raumstation angelegt hat. Darunter folgen nacheinander die noch verfügbare Energiemenge, die Ladung der Abwehrschirme, die noch vorhandenen Torpedos, die im Operationsgebiet der Enterprise insgesamt vorhandenen Klingonen, die zur Eingabe eines Befehls noch verfügbare Zeit und schließlich die Zeit in Sterntagen, die zur Vernichtung der Klingonen-Streitmacht insgesamt verbleibt.

Am unteren Bildschirmrand meldet sich der Bordcomputer der Enterprise mit Informationen über Vorgänge im Raumschiff, Fortgang des Kampfes usw. Rechts unten werden die Befehle des Kommandanten vor Eingabe angezeigt.

Im Verlauf des Spiels hat man verschiedene Möglichkeiten, auf die Aktionen des Raumschiffs Einfluß zu nehmen. Dazu nimmt der Bordcomputer entsprechende Befehle entgegen: Um die Enterprise zu bewegen, gibt man den Befehl FLUG ein und drückt ENTER (das muß man nach jeder Anweisung). Darauf fragt der Computer nach einem Kurs (dreistellige Zahl zwischen 000 und 359), der Flugweite (Anzahl der Sektoren) und der Antriebsart. Hier gibt es „Sprung" oder „Impuls", was man durch S oder I wählen muß. Die beiden Möglichkeiten unterscheiden sich dadurch, daß der Sprung mehr Energie verbraucht, der Impuls dafür mehr Zeit (Sterntag-Anteile). Außerdem muß ein Sprung mindestens über 8 Sektoren gehen (womit man immer in den nächsten Quadranten kommt). Gerät die Enterprise auf ihrer Fahrt über den Rand ihres Operationsgebiets, erfolgt entweder ein Sprung in eine zufällige Position, oder man landet im Hyperraum. Dann kann man einen Rücksprung durch Wahl einer Zahl zwischen 1 und 9 versuchen – mit etwas Glück kommt man daraufhin in den bekannten Weltraum zurück.

Die zweite Anweisung heißt SUCH und veranlaßt den Bordcomputer, die umliegenden Raumquadranten nach Sternen, Klingonen und gegebenenfalls einer Raumstation abzusuchen. Als dreistellige Zahlen wird das Ergebnis im Quadrantenfeld anstelle der XXXe oben am Bildschirm angezeigt. Dabei bedeutet die erste Ziffer die Anzahl der Sterne in einem Quadrant, die zweite die Anzahl der Raumstationen (nur 0 oder 1), die dritte die Klingonenzahl. Nach dem SUCH-Befehl kann man auf dem Quadrantenfeld auch die Enterprise-Position erkennen, weil das Raumschiff im Zentrum der abgesuchten Region steht. Um die Klingonen zu bekämpfen, was jeweils innerhalb eines Quadranten möglich ist, hat die Enterprise zwei Waffensysteme, Raumtorpedos und die Phaser-Strahlen. Diese Waffen aktiviert man mit den Befehlen TORP bzw. PHASER. Bei den Torpedos wird anschließend nach den Zielkoordinaten gefragt, die Phaser muß man mit einem Teil der vorhandenen Energiemenge laden. Der Abschuß erfolgt dann automatisch. Während die Torpedos das Ziel immer vernichten, können Klingonen einen Phaserangriff bei ungenügender Energiemenge überstehen und zurückschießen. Hier spielt die Entfernung zu dem Ziel bzw. den Zielen eine Rolle. Dann werden die Antriebe oder das Phasersystem teilweise

oder ganz außer Gefecht gesetzt, was dann im Schadensbericht (links auf dem Bildschirm) erscheint.

Davor kann man sich aber durch Laden der Abwehrschirme schützen, was durch die Anweisung SCHIRM und ebenfalls Angabe einer Energiemenge geschieht.

Schließlich kann man noch an eine Raumstation anlegen, wenn man das Raumschiff in einen benachbarten Raumsektor manövriert hat. Das geschieht durch den Befehl DOCK, allerdings nur, wenn zuvor die Schirme entladen wurden – durch SCHIRM und die Energieangabe 0. Im gedockten Zustand wird das Schiff mit Energie und Torpedos geladen und beschädigte Einrichtungen werden instand gesetzt.

Der Start des Raumschiffs zum Spielanfang geschieht durch den Befehl XX. Man hat dann insgesamt 30 Sterntage zur Verfügung sowie 10000 Einheiten Energie und 15 Torpedos. (Letztere lassen sich ja an den Raumstationen nachladen, doch die Zeit läuft unaufhaltsam weiter.) Auch muß jeder Befehl innerhalb einer bestimmten Zeitspanne erteilt werden. Jede Handlung des Spielers kostet Zeit und Energie, Bedienungsfehler werden meistens mit Zeitabzug bestraft. Wenn Zeit oder Energievorräte zu Ende sind, ehe alle Klingonen vernichtet werden konnten, gilt das Spiel als verloren.

Das Programm Startrek ist eines der längsten und kompliziertesten, die überhaupt in einem TRS-80 mit 16-K-Speicher Platz finden. Deshalb dürfte das Programmieren über die Tastatur des Computers sehr viel Ausdauer und Sorgfalt erfordern. In diesem Extremfall ist es vielleicht sogar ganz gut, wenn man nicht versucht, das ganze Programm auf einmal einzutippen, sondern etappenweise vorgeht und das Erreichte immer wieder auf eine Cassette überspielt. Zu leicht schleichen sich bei

nachlassender Konzentration Flüchtigkeitsfehler ein, die später sehr schwer zu finden sind.

Bemerkenswert an diesem Programm ist die Routine zur Übernahme von Eingaben durch den Spieler: Um die Zeit für eine Eingabe begrenzen zu können, aber trotzdem mit dem üblichen System mit der ENTER-Taste zu arbeiten, hat man die direkte Datenübernahme mit INKEY$ herangezogen, aber in einer Form, die das System mit ENTER nachahmt (Zeilen 4010 und 4015).

Eine letzte Bemerkung: Ursprünglich wurde das Programm in einer geringfügig anderen BASIC-Version geschrieben, als die Radio Shack-Computer jetzt verwenden. Deshalb findet man darin verschiedentlich den Ausdruck von der Art STRING$(30, CHR$(131)). Er ist gleichbedeutend mit STRING$(30,131) und kann beim Programmieren natürlich auch jeweils ersetzt werden. Daneben sind in Zeile 100 auch Matrizen mit kleinen Elementmengen definiert, was bekanntlich nicht erforderlich ist. Grundsätzlich soll allerdings an dieser Stelle ausdrücklich vor Änderungen gewarnt werden, die beim Einprogrammieren von Programmen vorgenommen werden, die als Listen vorliegen (wie in diesem Fall). Nur allzu oft stellt sich ein vermeintlich umständlicher oder unlogischer Programmteil hinterher als doch erforderlich heraus, und das „verbesserte" Programm läuft nicht. Wenn das Programm in der Originalform erst einmal funktioniert, bleibt noch genügend Gelegenheit zu Verbesserungen — bestimmt kann kein Programmierer von sich sagen, daß er immer Programme schreibt, an denen sich nichts mehr verbessern läßt und manche Dinge sind ja auch einfach eine Geschmacksfrage.

```
1 ' COPYRIGHT 1978  THE BOTTOM SHELF, INC.
2 ' ALL RIGHTS RESERVED  DO NOT COPY
3 ' P.O. BOX 49104  ATLANTA GA  30359
10 '  STAR TREK PHASE iii MARK I  (RAUMSCHIFF
   ENTERPRISE)
20 CLEAR750:
```

```
30 CLS:PRINT@512,CHR$(23)"INSTRUMENTE ANGESCHALTET":PRINT@64
0,"KOMMANDO-UEBERNAHME IN 4 SEK.
100 DIMG(8,8),G$(8,8),KL(3,1,1),SA$(8,8),Z$(20),Q(2),SR(2),S
E$(10,10):RL=851:RL$=CHR$(131)+CHR$(131)+CHR$(131)+CHR$(179)
110 S1$="####.##":S2$="###":S3$="#####":S4$="####":S5$="##":
S6$="####%":PS=0:IM=0:WA=0
120 BL$=STRING$(35," ")
130 ONERRORGOTO18000
1000 SD=RND(900)+RND(0)+2050:SE=SD:CN$="DOCK  ":EN=10002.31:
SH=0:TP=0:SW$=STRING$(24,CHR$(191)):SS$=STRING$(24,CHR$(170)
)
1998 GOTO2900
2000 FORX=0TO7:PRINT@340+X*64,SC$;:NEXTX:PRINT@394,QU(1);:PR
INT@398,QU(2);
2005 FORX=1TO8:FORY=1TO8:SE$(X,Y)="-":NEXTY,X
2010 FORX=1TOVAL(LEFT$(G$(QU(1),QU(2)),1))
2020 T1=RND(8):T2=RND(8):IFSE$(T1,T2)="-"THENSE$(T1,T2)="*"E
LSE2020
2030 NEXTX
2040 IFMID$(G$(QU(1),QU(2)),2,1)="0"THEN2060
2050 T1=RND(8):T2=RND(8):IFSE$(T1,T2)="-"THENSE$(T1,T2)="!"E
LSE2050
2060 IFRIGHT$(G$(QU(1),QU(2)),1)="0"THENPRINT@432,"GRUEN ";:
GOTO 2080
2065 PRINT@432,"ROT   ";
2070 FORX=1TOVAL(RIGHT$(G$(QU(1),QU(2)),1))
2075 T1=RND(8):T2=RND(8):IFSE$(T1,T2)="-"THENSE$(T1,T2)="K"E
LSE2075
2076 NEXTX
2080 FORX=0TO7:FORY=0TO7:PRINT@341+X*3+Y*64,SE$(X+1,Y+1);:NE
XTY,X:IFSR(1)=0GOTO2090
2085 IFSE$(SR(1),SR(2))="-"THENSE$(SR(1),SR(2))="E":GOTO2095
2090 T1=RND(8):T2=RND(8):IFSE$(T1,T2)="-"THENSE$(T1,T2)="E"E
LSE2090:SR(1)=T1:SR(2)=T2
2093 SR(1)=T1:SR(2)=T2
2095 PRINT@341+(SR(1)-1)*3+(SR(2)-1)*64,"E";
2098 PRINT@458,SR(1);:PRINT@462,SR(2);
2100 RETURN
2900 FORX=1TO5:Z$(X)="30":Z$(X+10)="0":NEXTX:Z$(6)="10":Z$(7
)="20":Z$(8)="30":Z$(9)="40":Z$(10)="50":Z$(15)="1":Z$(16)="
2":Z$(17)="3":Z$(18)="0":Z$(19)="0":Z$(20)="0"
2901 GOTO11000
3000 FORX=1TO8:FORY=1TO8:T=RND(10):T1=RND(10)+10:G$(X,Y)=Z$(
```

```
T)+Z$(T1):K=VAL(RIGHT$(G$(X,Y),1))+K:PRINT@687,K;:NEXTY,X
31ØØ FORZ=1TO2:T1=RND(8):T2=RND(8):G$(T1,T2)=LEFT$(G$(T1,T2)
,1)+"1"+RIGHT$(G$(T1,T2),1):NEXTZ
36Ø5 QU(1)=RND(8):QU(2)=RND(8)
36Ø7 FORX=ØTO2:PRINT@587+64*X,"";:PRINTUSINGS6$;Ø;:NEXTX:TP=
15:PRINT@623,TP;
361Ø GOSUB2ØØØ:GOTO5ØØØ
4ØØØ V=949:TC=1ØØ:QA$=CHR$(13)
4ØØ5 N=Ø:A$=INKEY$:A$="":A1$=""
4Ø1Ø PRINT@751,TC;" ";:TC=TC-1:IFTC=-1THEN45ØØELSEA$=INKEY$:
PRINT@V+N," ";:GOSUB42ØØ:PRINT@V+N,CHR$(95);:IFA$=""GOTO4Ø1Ø
4Ø15 IFA$=CHR$(8)THENA$="":IFN>ØTHENN=N-1:A1$=LEFT$(A1$,N):P
RINT@V+N,"  ":GOTO4Ø1Ø
4Ø2Ø IFA$=QA$THENGOTO45ØØ
4Ø25 PRINT@V+N,A$;:N=N+1:IFN=>9THENPRINT@V,STRING$(1Ø," ")EL
SE4Ø3Ø
4Ø27 GOTO4ØØ5
4Ø3Ø A1$=A1$+A$:A$="":GOTO4Ø1Ø
42ØØ PRINT@RL,RL$;:PRINT@875,CHR$(131);:RL=RL+3:IFRL=>875THE
NRL=851
42Ø1 RETURN
45ØØ PRINT@948,":           ";:PRINT@752,"    ";:PRINT@851,STR
ING$(25,CHR$(131));:EN=EN-2.31:GOSUB1Ø5ØØ:PRINT@752,"XXX";:R
ETURN
46ØØ PRINT@9Ø6,BL$;:PRINT@96Ø,STRING$(6Ø," ");:RETURN
49ØØ PRINT@9Ø6,BL$;:PRINT@96Ø,STRING$(6Ø," ");:IFA1$=""THENP
RINT@9Ø6,"REAKTIONSZEIT ZAEHLT, KAPITAEN!";:PRINT@96Ø,"IHR Z
OEGERN KOSTET EINEN STERNTAG.";:GOSUB19Ø1Ø:SD=SD-1:GOSUB495Ø
:RETURN
491Ø PRINT@9Ø6,"- - BEDIENUNGSFEHLER - -";:PRINT@96Ø,"DAS KO
STET EINEN STERNTAG.";:GOSUB19Ø1Ø:SD=SD-1:GOSUB495Ø:RETURN
495Ø IFSE=>SD+3ØGOTO2ØØØØ
4955 PRINT@368,"";:PRINTUSINGS1$;SE+(SE-SD);
496Ø PRINT@814,"";:PRINTUSINGS1$;3Ø-(SE-SD);
4999 RETURN
5ØØØ GOSUB46ØØ:PRINT@9Ø6," ERWARTE ANWEISUNG";:GOSUB4ØØØ
5ØØ2 IFWA>=1ØØTHENWA=1ØØ
5ØØ4 IFIM>=1ØØTHENIM=1ØØ
5ØØ6 IFPS>=1ØØTHENPS=1ØØ
5Ø1Ø PRINT@587,"";:PRINTUSINGS6$;WA;:PRINT@651,"";:PRINTUSIN
GS6$;IM;:PRINT@715,"";:PRINTUSINGS6$;PS;
5Ø2Ø IFA1$="FLUG"GOTO6ØØØ
5Ø3Ø IFA1$="TORP"GOTO7ØØØ
```

```
5040 IFA1$="PHASER"GOTO10000
5050 IFA1$="SCHIRM"GOTO8000
5060 IFA1$="SUCH"GOTO9000
5080 IFA1$="DOCK"GOTO13000
5200 PRINT@587,"";:PRINTUSINGS6$;WA;:PRINT@651,"";:PRINTUSIN
GS6$;IM;:PRINT@715,"";:PRINTUSINGS6$;PS;
5980 QP(1)=0:QP(2)=0:KF=0
5990 GOSUB4900:GOTO5000
6000 I=0:W=0:MF=0:GOSUB4600:PRINT@906,"DREISTELLIGE KURSZAHL
?";:GOSUB4000:IFLEN(A1$)<>3THENGOSUB4900:GOTO6000
6010 CS=VAL(A1$):IFCS>360THENPRINT@960,BL$;:PRINT@960,"KLEIN
ER ALS 360, BITTE!";:GOSUB19000:GOTO6000
6040 PRINT@333,A1$;:GOSUB4600:PRINT@906,"FLUGSTRECKE (SEKTOR
EN)?";:GOSUB4000:DS=VAL(A1$):IFDS>64PRINT@960,"DAMIT VERLASS
EN SIE DIE GALAXIE.";:GOSUB19000:GOSUB4600:PRINT@906,"COMPUT
ER WEIST BEFEHL ZURUECK.";:GOSUB19000
6041 IFDS<1THENA1$="":GOSUB4900:GOTO6040
6044 IFDS>64GOTO6047
6045 PRINT@960,"SPOCK AN KIRK: FUEHLEN SIE SICH IN ORDNUNG?"
;:GOSUB19000:GOTO5000
6047 IFMF=1THEN6050ELSEGOSUB4600:PRINT@906,"SPRUNG ODER IMPU
LS (S/I)?";:GOSUB4000:IFLEFT$(A1$,1)="S"THENW=1ELSEIFLEFT$(A
1$,1)="I"THENI=1ELSEPRINT@960,"WENN SIE NICHT WAEHLEN, FLIEG
EN WIR EBEN NICHT!";:GOSUB19010:GOTO5000
6048 IFLEFT$(A$,1)="S"THENDS=DS*(100-WA)/100
6049 IFLEFT$(A$,1)="I"THENDS=DS*(100-WA)/100
6050 GOSUB4600:PRINT@960,"MOTOREN GESTARTET";:GOSUB19000:DS=
DS/8:X1=QU(1)+(SR(1)-1)/8:Y1=QU(2)+(SR(2)-1)/8:QO(1)=QU(1):Q
O(2)=QU(2):SO(1)=SR(1):SO(2)=SR(2):CO=CS
6055 IFW=1ANDDS>=1THENEN=EN-DS*400:GOSUB10500:SD=SD-DS/8:GOS
UB4950:PRINT@777," SPRUNG";
6056 IFI=1THENSD=SD-DS/2:GOSUB4950:EN=EN-DS*80:GOSUB10500:PR
INT@777,STRING$(6," ")STRING$(6,CHR$(8))" IMPULS";
6057 IFW=1ANDDS<1THENSD=SD-1:EN=EN-100:GOSUB10500:GOSUB4600:
PRINT@906,"WAHL FEHLERHAFT";:GOSUB19000:GOTO5000
6065 CS=CS*.01745329:X2=SIN(CS)*DS:Y2=-COS(CS)*DS
6067 X2=X1+X2:Y2=Y1+Y2
6068 IFX2>=9THENX2=RND(7)+RND(0):Y2=RND(7)+RND(0):GOSUB4600:
PRINT@960,"******        RAUMSPRUNG        ******";:GOSUB19000
6069 IFY2>=9THENY2=RND(7)+RND(0):X2=RND(7)+RND(0):GOSUB4600:
PRINT@960,"******        RAUMSPRUNG        ******";:GOSUB19000
6070 QU(1)=INT(X2):SR(1)=INT((X2-QU(1))*8+1):QU(2)=INT(Y2):S
R(2)=INT((Y2-QU(2))*8+1)
```

```
6Ø71 IFQU(1)<=ØORQU(2)<=ØTHENPRINT@96Ø,"SCHIFF IM HYPERRAUM"
;:GOSUB19ØØØ:PRINT@9Ø6,"WEITE DES RUECKSPRUNGS?";:GOSUB4ØØØ:
QU(1)=QU(1)+VAL(A1$):QU(2)=VAL(A1$)+QU(2):IFQU(1)<=ØORQU(2)=
<ØTHEN2ØØØØ
6Ø75 IFQU(1)>=9ORQU(2)>=9THEN2ØØØØ
6Ø8Ø PRINT@394,QU(1);:PRINT@398,QU(2);:PRINT@458,SR(1);:PRIN
T@462,SR(2);
6Ø9Ø IFQU(1)<>QO(1)ORQU(2)<>QO(2)THENGOSUB2ØØØ:GOTO5ØØØ
6Ø92 IFSE$(SR(1),SR(2))<>"-"ORSE$(SR(1),SR(2))="+"THENQU(1)=
QO(1):QU(2)=QO(2):SR(1)=SO(1):SR(2)=SO(2):DS=DS*8-1:CS=CO:MF
=1:GOTO6Ø47
61ØØ SE$(SR(1),SR(2))="E":SE$(SO(1),SO(2))="-":PRINT@274+SR(
1)*3+SR(2)*64,"E";:PRINT@274+SO(1)*3+SO(2)*64,"-";
6999 GOTO5ØØØ
7ØØØ GOSUB46ØØ:PRINT@9Ø6,"POSITION DES ZIELS (X,Y)?";:GOSUB4
ØØØ:KS(1)=VAL(LEFT$(A1$,1)):KS(2)=VAL(RIGHT$(A1$,1)):IFSE$(K
S(1),KS(2))="K"THEN:GOSUB46ØØ:GOTO7ØØ5
7ØØ1 IFSR(1)=KS(1)ANDSR(2)=KS(2)THENCLS:FORX=1TO16:PRINTSTRI
NG$(63,CHR$(191)):NEXT:PRINT@522,CHR$(23);"AUF WIEDERSEHEN !
 !";:GOSUB19Ø1Ø:CLS:FORX=1TO1ØØ:PRINTSTRING$(63,CHR$(15Ø)):N
EXTX:GOTO0 20
7ØØ4 GOSUB77ØØ:GOTO7Ø95
7ØØ5 GOSUB46ØØ:PRINT@9Ø6,"ZIEL REGISTRIERT";:GOSUB19Ø1Ø:SD=S
D-.1:GOSUB495Ø:PRINT@96Ø,"TORPEDOS FERTIG ZUM FEUERN";:GOSUB
19Ø1Ø:IFTP<1THENGOSUB46ØØ:PRINT@9Ø6,"UEBERPRUEFUNG LAEUFT";:
GOSUB19Ø1Ø:PRINT@96Ø,"KEINE TORPEDOS VORHANDEN";:GOSUB19ØØØ:
GOTO5ØØØ
7Ø9Ø IFTP>=1THENTP=TP-1:PRINT@623,TP;
7Ø95 GOSUB46ØØ:PRINT@96Ø,"TORPEDOS ABGEFEUERT!";:GOSUB75ØØ
7Ø96 IFSE$(KS(1),KS(2))<>"K"THENGOSUB7ØØØ:GOSUB72ØØ:GOSUB73Ø
Ø:GOSUB711Ø:GOTO782Ø
7Ø97 GOSUB72ØØ:GOSUB73ØØ:GOSUB71ØØ:GOSUB712Ø:GOTO788Ø
71ØØ G$(QU(1),QU(2))=LEFT$(G$(QU(1),QU(2)),2)+RIGHT$(STR$(VA
L(RIGHT$(G$(QU(1),QU(2)),1))-1),1)
711Ø IFQU(2)=<4THENPRINT@(QU(1)-1)*4+(QU(2)-1)*64,G$(QU(1),Q
U(2));:RETURN
7115 IFQU(2)=<8THENPRINT@33+(QU(1)-1)*4+(QU(2)-5)*64,G$(QU(1
),QU(2));:RETURN
712Ø GOSUB46ØØ:PRINT@96Ø,"KLINGON VERNICHTET";:GOSUB19ØØØ:K=
K-1:PRINT@688,STRING$(3," ")STRING$(4,CHR$(8))K;
7125 IFRIGHT$(G$(QU(1),QU(2)),1)="Ø"THENPRINT@432,"GRUEN  ";:
713Ø RETURN
72ØØ SE$(KS(1),KS(2))="+":RETURN
```

```
7300 FORX=1TO8:FORY=1TO8:PRINT@274+X*3+Y*64,SE$(X,Y);:NEXTY,
X:RETURN
7500 FORX=1TO5:FORY=7TO0STEP-1:PRINT@340+Y*64,SS$;:NEXTY:FOR
Z=7TO0STEP-1:PRINT@340+Z*64,SC$;:NEXTZ,X:RETURN
7700 IFSE$(KS(1),KS(2))="!"THENGOSUB4600:PRINT@960,"AUTOMATI
SCHER TORPEDO-ABSCHUSS LAEUFT";:GOSUB19000:GOSUB4600:PRINT@9
60,"DAS WAR EIN STUETZPUNKT ! ! !";:GOSUB19010:RETURN
7710 IFSE$(KS(1),KS(2))="*"THENGOSUB4600:PRINT@960,"AUTOMATI
SCHER TORPEDO-ABSCHUSS LAEUFT";:GOSUB19010:GOSUB4600:PRINT@9
60,"DAS WAR EIN STERN.";:GOSUB19010:RETURN
7720 RETURN
7800 IFSE$(KS(1),KS(2))="!"ORSE$(KS(1),KS(2))="*"THENQ$=G$(Q
U(1),QU(2)):IFSE$(KS(1),KS(2))="!"THENQ$=LEFT$(Q$,1)+"0"+RIG
HT$(Q$,1):G$(QU(1),QU(2))=Q$
7810 IFSE$(KS(1),KS(2))="*"THENQ=VAL(LEFT$(Q$,1)):Q=Q-1:Q$=R
IGHT$(STR$(Q),1)+MID$(Q$,2,2):G$(QU(1),QU(2))=Q$
7815 IFSE$(KS(1),KS(2))="-"ORSE$(KS(1),KS(2))="+"THENGOSUB46
00:PRINT@960,"DAS WAR WOHL NICHTS!";:GOSUB19010
7816 GOSUB7110:RETURN
7820 GOSUB4600:X=RND(5):ONXGOTO7830,7840,7850,7860,20000
7830 PRINT@960,"DAS WAR NICHT SEHR SCHLAU!";:GOTO7870
7840 PRINT@960,"DAS WAR SEHR SCHLAU ! !";:GOTO7870
7850 PRINT@960,"KLINGONEN SOLLST DU ABSCHIESSEN!";:GOTO7870
7860 PRINT@960,"LASS´ DIR DEIN LEHRGELD WIEDERGEBEN!";
7870 GOSUB19010:GOSUB10500:R=RND(25):SD=SD-1:GOSUB4950:TP=TP
-1:PRINT@623,STRING$(3," ");STRING$(3,CHR$(8));TP;
7880 R=RND(25):IFR=10THENWA=WA+RND(10)+10
7890 IFR=20THEN1M=IM+RND(10)+10
7900 SD=SD-.21:GOSUB4950
7999 GOTO5000
8000 EN=EN+SH:PRINT@496,STRING$(7," ");:PRINT@495,EN;:SH=0:P
RINT@560,STRING$(7," ");:GOSUB4600:PRINT@906,"SCHIRME LADEN"
;:GOSUB4000:IFEN>VAL(A1$)THENSH=VAL(A1$)ELSEGOSUB4600:PRINT@
960,"VORRAT ZUR ZEIT UNZUREICHEND";:GOSUB19000:GOTO8000
8005 EN=EN-SH:PRINT@496,STRING$(7," ");:PRINT@495,EN;
8010 PRINT@560,STRING$(7," ");:PRINT@559,SH;
8020 GOTO5000
9000 GOSUB4600:PRINT@906,"SUCHE LAEUFT";:FORX=QU(1)-2TOQU(1)
:FORY=QU(2)-2TOQU(2):IFX=>8ORX=<-1THENGOTO9020
9005 IFY=<3ANDY=>0THENPRINT@X*4+Y*64,G$(X+1,Y+1);:GOTO9020
9010 IFY=>4ANDY=<7PRINT@33+X*4+(Y-4)*64,G$(X+1,Y+1);
9020 NEXTY,X
9030 IFY=2ORY=9ORX=2ORX=9THENGOSUB4600:PRINT@960,"AN DER GRE
```

```
NZE DES BEKANNTEN WELTRAUMS";:GOSUB19ØØØ:X=RND(15):IFX=3THEN
PRINT" WO NOCH NIEMAND WAR";:GOSUB19Ø3Ø
9Ø4Ø SD=SD-.13:GOSUB495Ø:EN=EN-.96:GOSUB1Ø5ØØ
91ØØ GOTO5ØØØ
1ØØØØ KF=Ø:GOSUB46ØØ:PRINT@96Ø,"PHASER FEUERBEREIT";:IFQP(1)
=QU(1)ANDQP(2)=QU(2)THENKF=1
1ØØØ5 IFPS=>1ØØTHENGOSUB46ØØ:PRINT@96Ø,"PHASER AUSGEFALLEN";
:GOSUB19Ø1Ø:GOTO5ØØØ
1ØØ1Ø PRINT@9Ø6,"PHASER LADEN";:GOSUB4ØØØ:IFVAL(A1$)>ENTHENG
OSUB46ØØ:PRINT@96Ø,"VORRAT ZUR ZEIT UNZUREICHEND";:PRINT@9Ø6
,"NUR NOCH"EN"EINHEITEN";:GOSUB19ØØØ:GOTO5ØØØ
1ØØ15 IFA1$=""THENGOSUB46ØØ:PRINT@96Ø,"SPOCK AN KAPITAEN: SC
HNELL!";:GOSUB19ØØØ:GOTO5ØØØ
1ØØ2Ø IFVAL(A1$)<1ØØTHENGOSUB46ØØ:PRINT@9Ø6,"FEHLENDE ANTWOR
T OHNE FOLGEN";:GOSUB19ØØØ:PRINT@96Ø,"SPOCK AN KIRK:   SOLL
ICH SIE ABLOESEN?";:GOSUB19ØØØ:GOTO5ØØØ
1ØØ25 EN=EN-VAL(A1$):PRINT@495,STRING$(7," ")STRING$(7,CHR$(
8))EN;:PP=VAL(A1$)*(1-PS/1ØØ)
1ØØ3Ø Z=Ø:KP=Ø:FORX=1TO3:KP(X)=Ø:NEXTX
1ØØ35 GOSUB46ØØ:PRINT@96Ø,"KLINGON-BERICHT:          NUMMER
  =";:PRINT@996,"      Ø";
1ØØ4Ø FORX=1TO8:FORY=1TO8:IFSE$(X,Y)<>"K"THEN1Ø1ØØ
1ØØ5Ø KP=KP+1:KP(KP)=SQR((SR(1)-X)[2+(SR(2)-Y)[2)
1ØØ6Ø KL(KP,1,1)=Y:KL(KP,Ø,Ø)=X
1ØØ7Ø KP(KP)=1/(KP(KP)[2)
1ØØ8Ø KL(KP,Ø,1)=KP(KP)
1ØØ9Ø KL(KP,1,Ø)=2ØØ
1ØØ95 PRINT@998,"   "KP;:GOSUB19Ø1Ø
1Ø1ØØ NEXTY,X
1Ø11Ø QP(1)=QU(1):QP(2)=QU(2)
1Ø19Ø GOSUB46ØØ:PRINT@96Ø,"PHASER FEUERN";
1Ø195 Y=Ø
1Ø2ØØ Y=Y+1 :FORX=7TOØSTEP-1:PRINT@34Ø+64*X,SW$;:PRINT@34Ø+6
4*X,SC$;:NEXTX
1Ø2Ø1 IFY<>5GOTO1Ø2ØØ
1Ø2Ø2 Y=Ø:IFKP=ØTHEN1Ø335
1Ø31Ø FORX=1TOKP:KL(X,1,Ø)=KL(X,1,Ø)-PP*KP(X)
1Ø32Ø IFKL(X,1,Ø)<=ØTHENPRINT@274+3*KL(X,Ø,Ø)+64*KL(X,1,1),"
+";:GOSUB71ØØ:SE$(KL(X,Ø,Ø),KL(X,1,1))="+":GOSUB712Ø:KP=KP-1
1Ø33Ø NEXTX
1Ø332 SH=SH-(KP[2)*2ØØ*(KP(1)+KP(2)+KP(3)):PRINT@559,STRING$
(7," ")STRING$(7,CHR$(8))SH;:IFSH<ØTHENPS=1ØØ-PS*ABS(SH/2ØØ)
:PRINT@715,"";:PRINTUSINGS6$;PS;:PRINT@56Ø,STRING$(7," ")STR
```

```
ING$(7,CHR$(8))"0000";:SH=0
10335 GOSUB7300
10340 R=RND(25):IFR=10THENWA=WA+RND(10)+10
10350 IFR=20THENIM=IM+RND(10)+10
10499 GOTO5000
10500 EG=21
10510 IFEN=<0THENCLS:PRINTCHR$(23):EG$="ENERGIE VERBRAUCHT..
.":PRINT@512,RIGHT$(EG$,EG):EG=EG-1:GOSUB19010:IFEG>0THEN105
10ELSE20000
10520 PRINT@496,STRING$(7," ");:PRINT@495,EN;:IFEN<500,GOSUB
4600:PRINT@960,"ENERGIEVORRAT GEFAEHRLICH NIEDRIG - SCHIRME
ENTLADEN?";:GOSUB19000
10525 RETURN
11000 GA$="XXX XXX XXX XXX XXX XXX XXX XXX  XXX XXX XXX XXX
XXX XXX XXX XXX":CO$="KURS             XXX":QU$="QUADRANT   X -
X":SE$="SEKTOR      X - X":DR$="SCHADENSBERICHT":WA$=CHR$(149
)+"SPRUNG         XXX"
11010 IM$=CHR$(149)+"IMPULS        XXX":PH$=CHR$(133)+"PHASER
      XXX":ST$="XXXX.XX STERNTAG":CN$="XXXXXX   ZUSTAND":EN$=
"XXXXX    ENERGIE":SH$="XXXX      SCHIRME":TP$="XX        TORPE
DOS":KL$="XX      KLINGONEN"
11020 SC$=" -  -  -  -  -  -  -  - ":FT$=STRING$(25,CHR$(140
)):FB$=STRING$(26,CHR$(131)):CC$="COMPUTER:
                         BEFEHL:":TC$=" REST-ZEIT":DV$="ANTRIEB"
11030 CLS:FORX=0TO3:PRINT@32+64*X,CHR$(149);:NEXTX:PRINT@275
,FT$;:SET(64,12):PRINT@851,FB$;:PRINT@275,CHR$(188);:PRINT@3
00,CHR$(188);:FORX=0TO7:PRINT@339+64*X,CHR$(191);:PRINT@364+
64*X,CHR$(191);:NEXTX
11040 FORX=0TO3:PRINT@0+64*X,GA$;:PRINT@64*X+32,CHR$(149);:N
EXTX:SET(64,12):PRINT@320,CO$;:PRINT@384,QU$;:PRINT@448,SE$;
:PRINT@512,DR$;:PRINT@576,WA$;:PRINT@640,IM$;:PRINT@704,PH$;
:PRINT@768,DV$;
11050 PRINT@368,ST$;:PRINT@432,CN$;:PRINT@496,EN$;:PRINT@560
,SH$;:PRINT@624,TP$;:PRINT@688,KL$;:PRINT@758,TC$;:FORX=0TO7
:PRINT@340+64*X,SC$;:NEXTX:PRINT@816,"XX.XX     ST REST";:PRI
NT@896,CC$;
11060 PRINT@906,BL$;:PRINT@906,"STARTBEFEHL: 'XX'";:PRINT@96
0,BL$;:PRINT@968,"ENTERPRISE STARTBEREIT";:GOSUB4000:IFA1$<>
"XX"THENGOSUB4900:GOTO11060
11070 PRINT@906,BL$;:PRINT@906,"COMPUTER IN BETRIEB";:PRINT@
965,BL$;:PRINT@965,"WARTEN - UEBERPRUEFUNG LAEUFT";
11100 PRINT@368,"";:PRINTUSINGS1$;SD;:PRINT@432,"DOCK  ";:PR
INT@495,EN;:GOTO3000
```

```
12ØØØ PRINT@587,"";:PRINTUSINGS6$;WA;:PRINT@651,"";:PRINTUSI
NGS6$;IM;:PRINT@715,"";:PRINTUSINGS6$;PS;
121ØØ GOTO5ØØØ
13ØØØ FORX=SR(1)-1TOSR(1)+1:FORY=SR(2)-1TOSR(2)+1:D$=SE$(X,Y
):IFD$="!"THEND1$="DOCK  "
13ØØ5 NEXTY,X:IFD1$="DOCK  "GOTO13Ø2Ø
13Ø1Ø GOSUB46ØØ:PRINT@96Ø,"MANOEVER MISSLUNGEN, POSITION FAL
SCH";:GOSUB19ØØØ:GOSUB46ØØ:PRINT@96Ø,"DADURCH GEHT EIN STERN
TAG VERLOREN.";:GOSUB19ØØØ:SD=SD-1:GOSUB495Ø:GOTO5ØØØ
13Ø2Ø GOSUB46ØØ:IFSH<>ØTHENPRINT@96Ø,"MANOEVER MISSLUNGEN, S
CHIRME NOCH GELADEN";:GOSUB19ØØØ:PRINT@96Ø,"FEHLERHAFTES VOR
GEHEN KOSTET EINEN STERNTAG.";:GOSUB19ØØØ:SD=SD-1:GOSUB495Ø:
GOTO5ØØØ
13Ø25 IF K<=Ø THEN 21ØØØ
13Ø3Ø PRINT@432,D1$;:D1$="":EN=1ØØØØ:TP=15:WA=Ø:IM=Ø:PS=Ø:GO
SUB46ØØ:PRINT@96Ø,"RAUMSCHIFF EINGEDOCKT, INSTANDSETZUNG LAE
UFT";:GOSUB19ØØØ:PRINT@495,EN;"  ";:PRINT@624,"";:PRINTUSING
S5$;TP;
13Ø4Ø PRINT@587,"";:PRINTUSINGS6$;WA;:PRINT@651,"";:PRINTUSI
NGS6$;IM;:PRINT@715,"";:PRINTUSINGS6$;PS;:SD=SD-.5:GOSUB495Ø
:PRINT@56Ø,"      ";:PRINT@559,SH;
131ØØ GOTO5ØØØ
18ØØØ ES=ERR/2+1:EE=ERL:PRINTES;:PRINT"    ";:PRINTEE;:ONERRO
RGOTO18ØØØ:STOP
19ØØØ FORXT=1TO1ØØØ:NEXTXT:RETURN
19Ø1Ø FORXT=1TO5ØØ:NEXTXT:RETURN
19Ø3Ø FORXT=1TO2ØØØ:NEXTXT:RETURN
2ØØØØ CLS:PRINTCHR$(23):PRINT@384,"DIE GALAXIE IST VERLOREN!
":GOSUB19ØØØ:GOSUB19ØØØ:CLS:PRINT@512,".....UND DU WIRST NIC
HT ZUM ADMIRAL BEFOERDERT!":GOSUB19Ø3Ø:GOTO1
21ØØØ CLS:PRINTCHR$(23):PRINT@512,"DU HAST DIE GALAXIE GERET
TET ! ! !";:GOSUB19Ø3Ø
21Ø1Ø CLS:PRINT@512,"DEINE BEFOERDERUNG ZUM ADMIRAL IST AUFG
ESCHOBEN WORDEN, BIS DU
21Ø2Ø PRINT"VON ARCTURUS IV ZURUECKKEHRST, DEREN GALAXIE DU
RETTEN MUST.
21Ø3Ø PRINT"DEIN RAUMSCHIFF IST REPARIERT, UND DU STARTEST..
.." END.
21Ø4Ø GOSUB19Ø3Ø:CLS:PRINTCHR$(23):PRINT@512,"        JDIM
MERGELISTMERGE@B@B@BDvu"br\\\ub
```

50000

Verlagsprogramm

Übersicht lieferbarer Bücher

Bestell Nr.	ISBN	Verfasser	Titel	Preis DM
1	3-921682-01-0	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 1	19,80
2	3-921682-02-9	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 2	19,80
3	3-921682-03-7	Gebauer	Elektronik im Auto	9,80
4	3-921682-04-5	Lorenz	IC-Handbuch, TTL, CMOS, Linear	19,80
5	3-921682-05-3	Steinbach	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	3-921682-06-1	Steinbach	IC-Schaltungen, TTL, CMOS, Linear	9,80
7	3-921682-33-9	Hofacker	Elektronik Schaltungen	5, –
8	3-921682-08-8	Lorenz	IC-Bauanleitungs-Handbuch	19,80
9	3-921682-09-6	Lorenz	Feldeffekttransistoren	5, –
10	3-921682-34-7	Lorenz	Elektronik und Radio, 4. Auflage	19,80
11	3-921682-11-7	Lorenz	IC-NF Verstärker	9,80
12	3-921682-12-6	Bernstein	Beispiele Integrierter Schaltungen (BIS)	19,80
13	3-921682-13-4	Lorenz	HEH, Hobby Elektronik Handbuch	9,80
14	3-921682-14-2	Lorenz	IC-Vergleichsliste	29,80
15	3-921682-15-0	Lorenz	Optoelektronik Handbuch	19,80
16	3-921682-16-9	Bernstein	CMOS Teil 1 Einführung Entwurf, Schaltbeispiele	19,80
17	3-921682-17-7	Bernstein	CMOS Teil 2 Entwurf und Schaltbeispiele	19,80
18	3-921682-18-5	Bernstein	CMOS Teil 3 Entwurf und Schaltbeispiele	19,80
19	3.921682-19-3	Lorenz	IC-Experimentier Handbuch	19,80
20	3-921682-20-7	Lorenz	Operationsverstärker	19,80
21	3-921682-21-5	Lorenz	Digitaltechnik Grundkurs	19,80
22	3-921682-22-3	Bernstein	Mikroprozessoren, Eigenschaften und Aufbau 2. Aufl.	19,80
23	3-921682-23-1	Lorenz	Elektronik Grundkrus Kurzlehrgang Elektronik	9,80
24	3-921682-35-5	Hans Peter Blomeyer-Bartenstein	Mikrocomputer Technik	29,80
25	3-921682-25-8	C. Lorenz	Hobby Computer Handbuch	29,80
26	3-921682-26-8	H. Bernstein	Mikroprozessor Teil 2	19,80
27	3-921682-27-4	C. Lorenz	Mikrocomputer Software Handbuch	29,80
28	3-921682-28-2	C. Lorenz	Lexikon + Wörterbuch für Elektronik und Mikroprozessortechnik LEM	29,80
29	3-921682-29-0	C. Lorenz	Mikrocomputer Datenbuch	49,80
30	3-921682-30-4	C. Lorenz	Aktivtraining-Mikrocomputer (Ordner)	49,80
31	3-921682-31-2	C. Lorenz	57 Programme in BASIC (Cames)	39, –
32	3-921682-32-0	C. Lorenz	Circuits Digital Et Pratique	19,80
33	3-921682-36-6	Dr. Hatzenbichler	Microcomputer Programmierbeispiele	19,80
41	–	–	Experimentierplatine für 14, 16, 24, 28 und 40 polige DIL IC's	79, –
105/1	–	–	TTL-Experimentierbuch	5, –
106/1	–	–	CMOS-Experimentierbuch	5, –
109	3-921682-43-6	P. Heuer	6502 Microcomputer Aufbau und Programmierung	29,80
110	3-921682-49-5	C. Lorenz	Programmierhandbuch für PET	29,80
113	3-921682-48-7	C. Lorenz	BASIC Programmierhandbuch	19,80

Ing. W. Hofacker GmbH Verlag
8 München 75

Inhaltsverzeichnis